# REAL-TIME RAILWAY SIMULATION IN JAVA

BY

Dominic James Ancheta

DESIGN PROJECT REPORT

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science
in Computer Engineering
in the School of Engineering of
Santa Clara University, 1998

Santa Clara, California

# TABLE OF CONTENTS

Page

Dominic James Ancheta

Department of Computer Engineering

Santa Clara University, 1998

## ABSTRACT

This document describes the design and usage of the recreational software application named ‘Real-time Railway Simulation in Java’. It is an application designed using object-oriented methods, relational database technology, and client-server technology over the Internet. The design section describes the design of the application using object-oriented design and relational database design reports and diagrams. The usage section shows users of the application how to run it and use its interface, as well as create his/her own custom railways. The design report is a complete view of the application, but it has not been totally implemented (approximately 80% complete) as of the time of this writing. So, the usage section only describes the working part of the application.

SENIOR PROJECT DESIGN REPORT

# Real-time Railway Simulation in Java

Revision 1.7

Authored by
Dominic Ancheta

June 8, 1998

Senior Thesis Advisor:
Dr. Linda Seiter

Santa Clara University

# Senior Project Design Report Table of Contents

# Document Revision History

| Date | Rev. No. | Changes made |
|---|---|---|
| ***Dec. 6, 1997*** | V1.0 | First release with preliminary design and diagrams |
| ***Jan. 3, 1998*** | V1.1 | Modification to relational database design |
| ***Jan. 23, 1998*** | V1.2 | Modifications applied after design evaluation before coding. Use triple buffering technique for animation. Use multiple train objects rather than multiple train threads to eliminate context-switching and other overhead. |
| ***Jan. 24, 1998 to Feb. 22, 1998*** | V1.3 To V1.5 | Iterative design changes while coding |
| ***Mar. 3, 1998*** | V1.6 | References to train threads eliminated. Conversion to just objects. Changed class diagrams and reduced classes used and messaging between them. |
| ***Apr. 8, 1998*** | V1.7 | Updated scenarios: Removed data entry scenarios.<br>Updated interaction diagrams to current implementation; specifically, the sections on railway movement.<br>Updated class diagrams<br>Updated class state diagrams<br>Added problem history documentation<br>E-R Diagram modified: 1) Added a wait_time and a move_delay time to the Arrival table. 2) Removed wait_time from Rail_Location table |

# SECTION 1

## Scope Analysis of Problem Space

This section describes the design of the Real-time Railway Simulation in detail.

It covers:

- A description of what the application is and what it does
- The user-interface.
- Identification of problems in designing the simulation.
- Technical details of the simulation in sentence format.

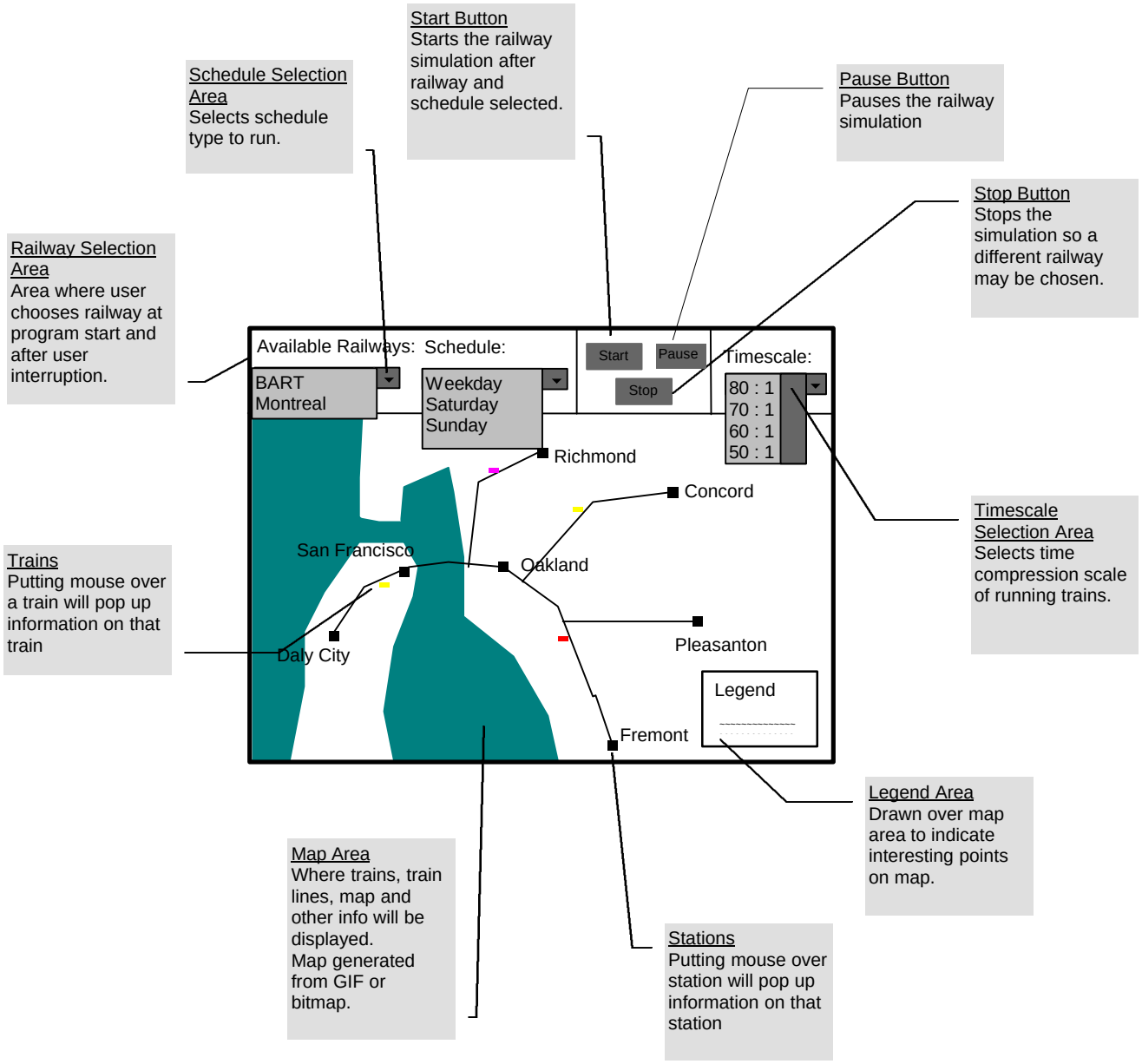



**Figure 1.1. Real-time Railway Simulation in Java GUI Interface**

### Railway Simulation Functionality

Above is the proposed GUI interface for the Real-time Railway Simulation. It will be designed to run over the Internet. Descriptions of the GUI elements are described in the callout boxes. The simulation will not run in an applet, but in a standalone application window.

When the user clicks start, following railway and schedule selection, data is loaded from a remote database into memory. Data includes railway locations, stations, train arrival times, train ids, etc.

For simplicity the trains travel only in straight lines. There can be direction changes between stations, however, so that travel from station to station is not always a direct line. The points where the train changes directions are what I call angle coordinates. Stations and angle coordinates are hereon known generally as railway locations.

Multiple independent trains will eventually be animated on screen simulating a railway system's train travels for one complete railway day depending on the schedule the user chooses (e.g. Weekday, Saturdays, Holidays, etc.).

The simulation ends when the simulation naturally comes to an end after the last train leaves the railway, or when the user presses the stop button.

**Map Format**
The simulation will use a bitmap created by Railway Simulation developers.

**Animation Technique**
Trains will be animated over the above-mentioned map, using the midpoint algorithm and use a triple buffering method for each frame of animation.

**Data Format**
Data on the coordinates and station arrival will be stored in a Microsoft Access relational database. The simulation will use JDBC to access the data. Data will be retrieved after a railway and schedule is chosen, and the start button is pressed. The database will not be accessed while the simulation is in running mode.

**Timing Method and Train Location Synchronization**
There will be a railway clock, which is initialized to the current PC time just before trains are to start animating. This will be used to synchronize with the arrival times of each train so that its current location can be determined. This will be used to keep track of the time. A time compression value will also be factored into this, since a simulation in real time would be slow and cumbersome to watch.

**Train Objects**
The current method of moving the trains is to have the simulation loop through each active train object when the simulation is running. The code will run sequentially for the current tick (which may vary depending on running speed) and calculate the location for this tick.

# SECTION 2

## Scenarios

This section describes the major scenarios the Real-time Railway Simulation in Java will execute while it is running.

**GUI Event Scenarios**

- Application is executed on web client: initial data information retrieved and GUI interface created, and wait for user event.
- User chooses a railway and schedule from railway combo list. (in stopped mode)
- Start button pressed.
  - Railway data corresponding to selection loaded into local memory. (This includes creation of Railway Location, Line, and Train objects and their initialization with coordinates, arrival times, IDs, etc.)
  - Railway simulation is initialized: includes PC clock initialization to earliest train departure that is stored in the Railway Data, and train object current coordinates initialized to their first station departure coordinates.
  - Railway bitmap loaded into buffer memory. (Once the data and bitmap are loaded, trains will run on screen until schedule ends or user ends simulation.)
- No railway selected when Start button pressed.
- User ends railway simulation by pushing stop button. (in running or paused mode)
- User quits railway simulation application by pushing quit button.(any mode)
- User chooses different time compression factor. (any mode)
- User scrolls simulation area and it is repainted. (in running and/or pause mode. May require separate cases.)
- Scenarios to implement if there is time:
  - User presses the Pause button. (in running mode)
  - User presses the Resume button. (in paused mode)
  - User minimizes window.
  - User maximizes window.
  - User resizes window.
  - User clicks cursor on train, which pops up train info box. (in running or paused mode)
  - User places mouse cursor over train station, which pops up station info box. (In running or paused mode)

**Trains running scenarios**

Simulation at this point is in running mode; trains are running. For each train object:

- Time for train to start arrives and it is drawn to screen for the first time, waiting at its first station.
- Train waits for a specified amount of time at a station. (This includes the first station a train begins from. The time given in a schedule is assumed to be the time the train opens its doors for its first passengers. This also includes the train arriving at final station in line and then disabled.)
- Train arrives at an angle coordinate and direction change calculations are made on route to station. Train drawn.
- Train starts moving after waiting at a train station and then is drawn to screen at the next coordinate.
- Train moves from one pixel to the next pixel en route to the next station, and is drawn to screen.
- Train has finished waiting at its last station and disappears from the simulation
- Final train in entire schedule arrives at last station; simulation ends and the simulation is put in stopped mode automatically.

For the trains running scenarios:

1. Stopped mode means the simulation is not running; at this point, the application waits for user to select a railway and schedule to run and push the start button (or exit simulation).
2. Running mode means the simulation is running; trains are running on the graphical simulation area.
3. Paused mode means the user pressed the pause button while the simulation is running. Trains in the graphical simulation area are suspended during this time.

Angle coordinates may be used between train stations. Since trains will often not move in straight lines all the time between stations, this angle coordinate will be used as a point for the train to change direction in the simulation screen.

**Exception Scenarios**

For the following scenarios, the application will display an error message window and the simulation fails to run.

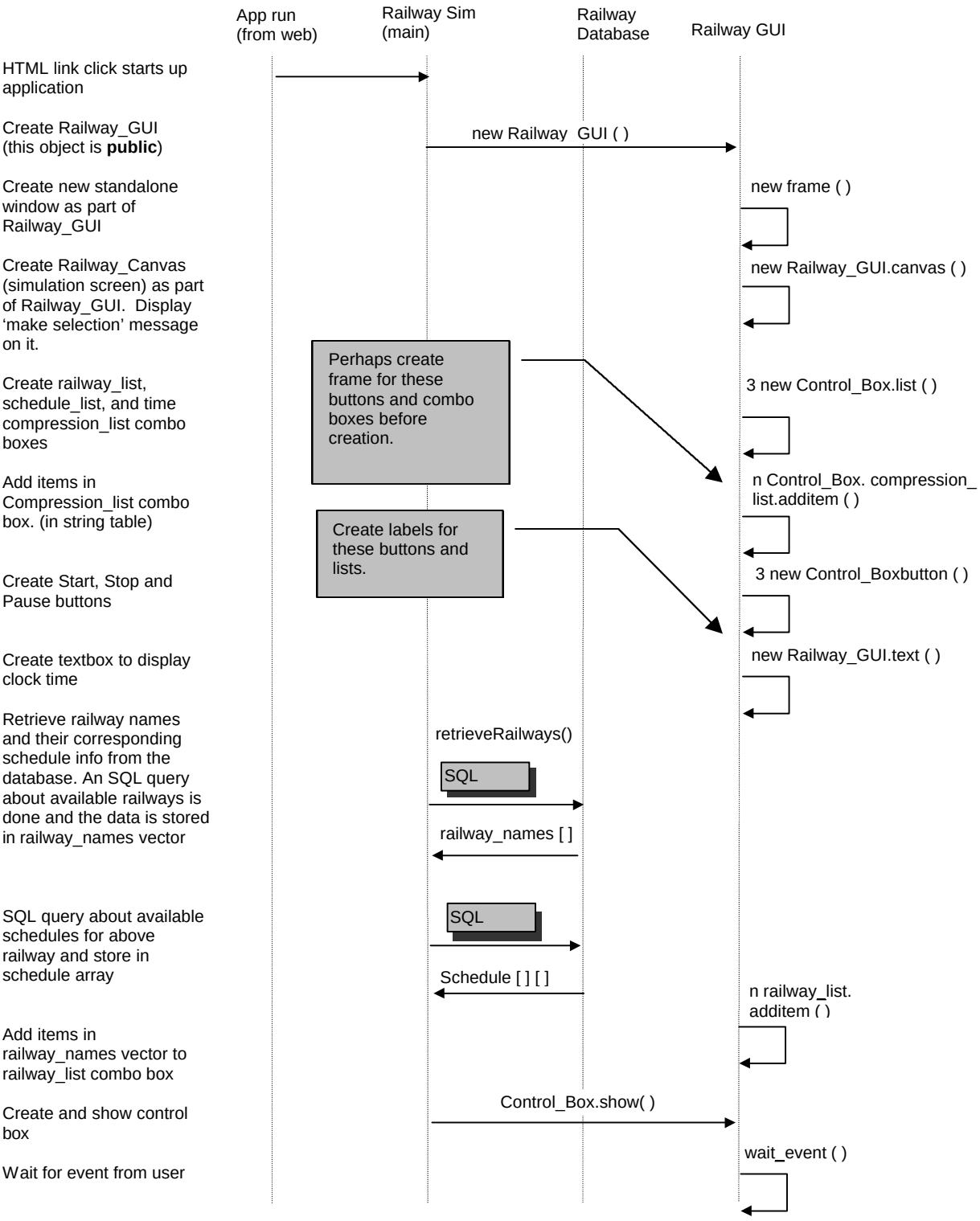
- After start button pressed, some schedule data is not in sequential order.
- After start button pressed, coordinates data for station(s) are not within simulation area boundaries.

# SECTION 3

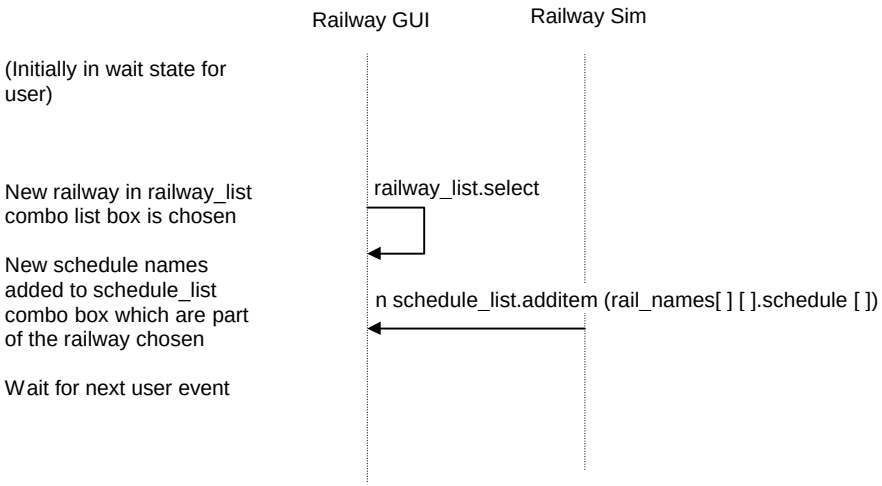
## Interaction Diagrams

This section shows the interaction of objects for the Real-time Railway Simulation project. An interaction diagram is created for each of the scenarios mentioned in Section 2. These will be used to identify, as well as to aid in discovering, new classes, methods and message passing in the design of the simulation.

**Application is executed on web client: initial data information retrieved and GUI interface created, then wait for user event**



**User chooses a railway from railway combo list** (in stopped mode)



**Start button pressed: Railway data corresponding to selection loaded (Rail Location, Line and Train objects) into memory**

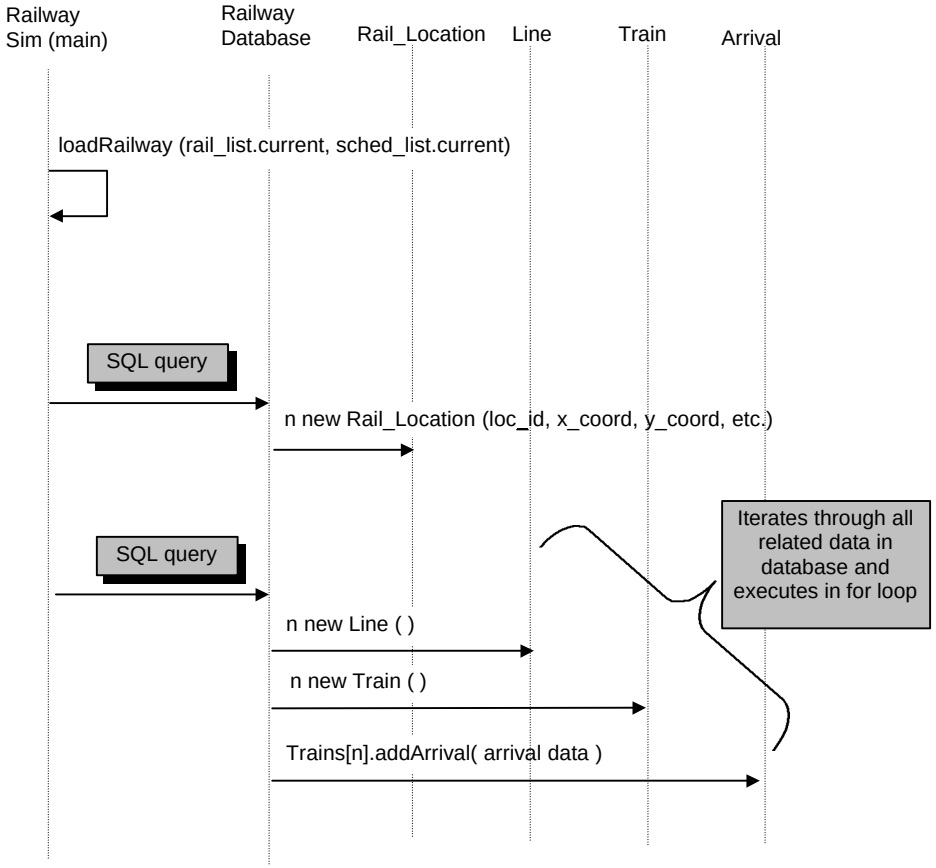
(Initially waiting for user event)

Start button pressed.

Create and allocate space for Rail Location, Line and Train objects, then load them with data associated with selections from railway and schedule combo lists as follows:

Perform SQL query to retrieve railway location information. Create station or angle coordinate array of objects and store info in there.

Perform SQL fetch or select routines with selections from above step AND
Retrieve line, railway, train id and train arrival information into line and train objects
(see class data structures shown below)

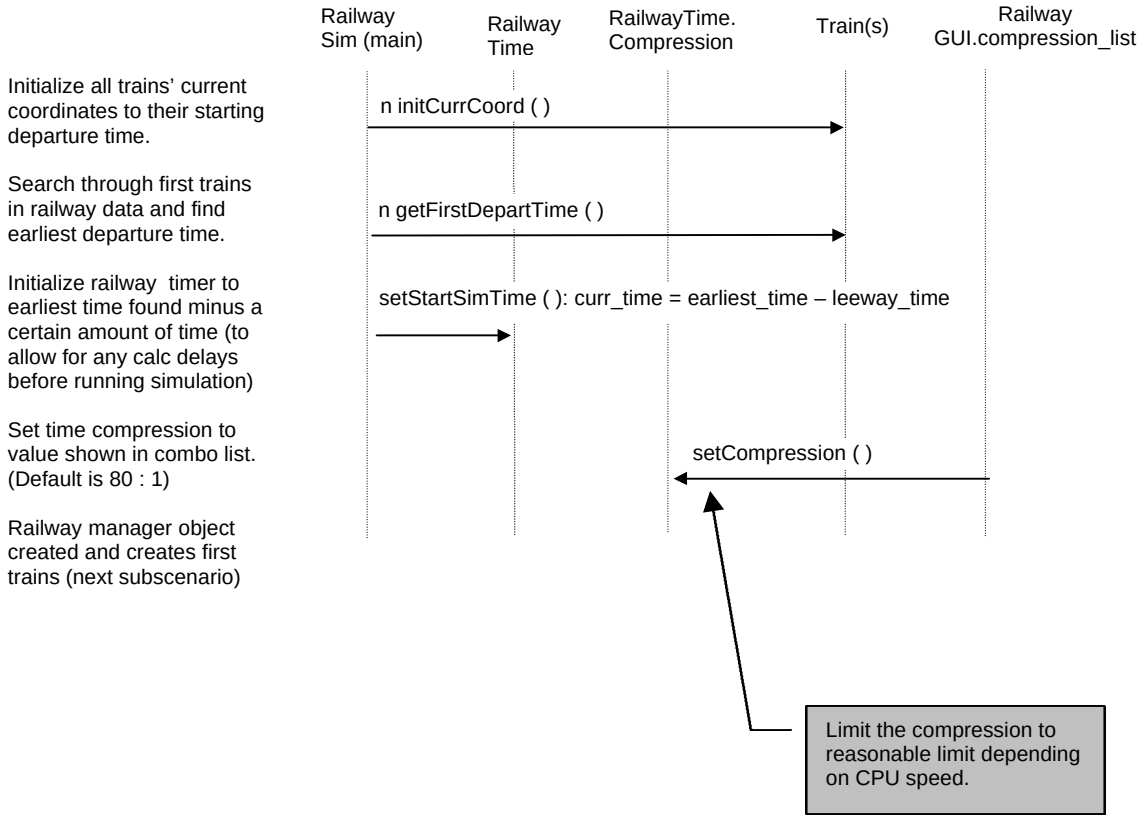



---

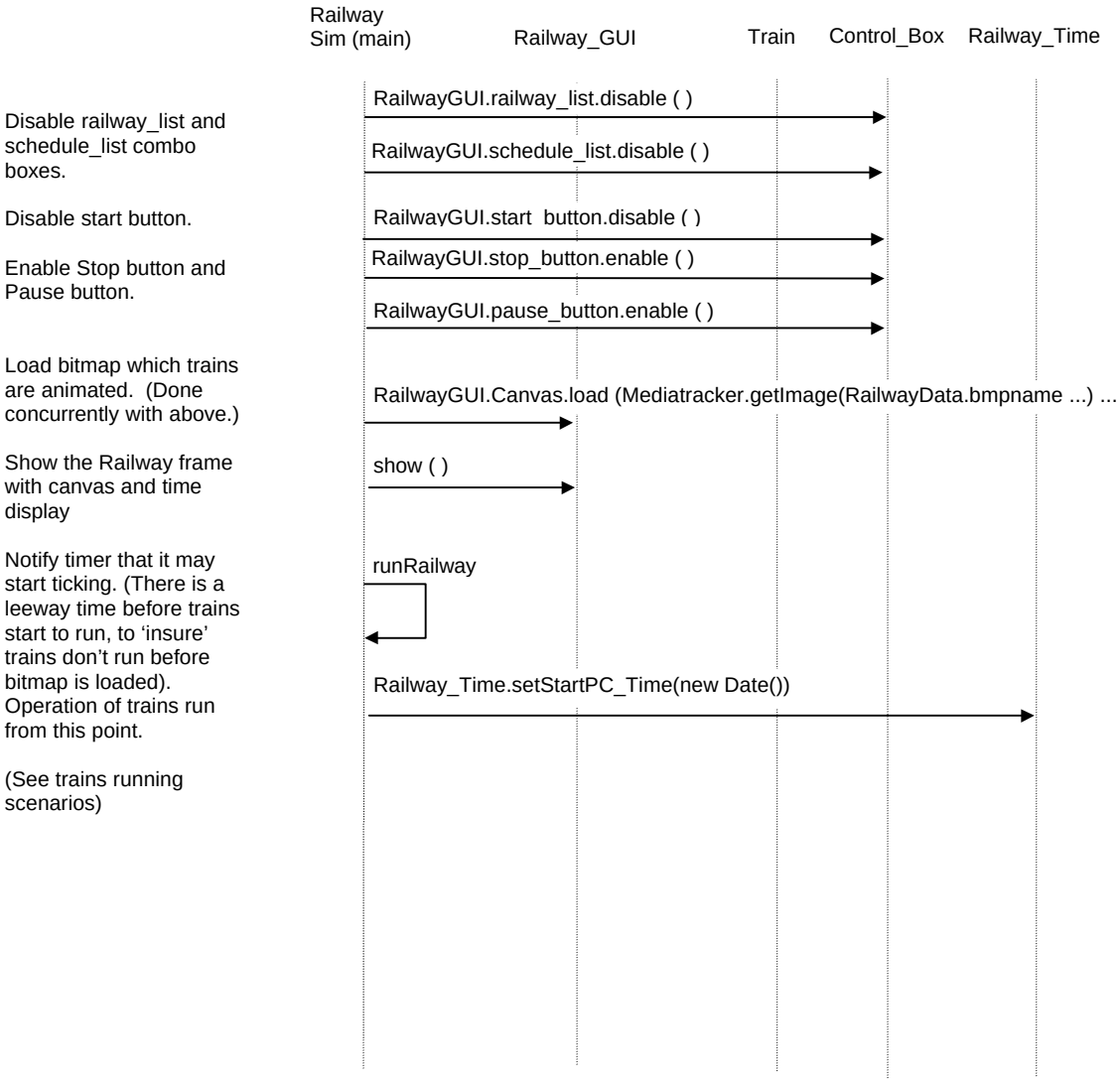
Line data structure format (part of RailwayData object):

```
String Railway_Name;
String Bitmap_Name;
Class Line [ ] {
        String Line_Name;
        String Schedule_Name;
        Color color;
}
Class Train {                       // Train class fields (part of Line data class)
        Train_state;                // train the state is in; one of NOT_STARTED, IS_WAITING, IS_MOVING,
                                    //  or IS_FINISHED
        Line_num_ref;               // line number this train is part of
        Arrival [ ];                // arrival times at each railway location
}
Class Railway_Loc [ ] {             // fields for Railway_Loc class
        String Name;                // name of this railway location
        Coord Rail_coord;           // railway location coordinates
}
Class Arrival [ ] {                 // this class contains holds information for arrival times at railway locations
                                    // assigned to trains
        Time arrival_time;          // time train is expected to arrrive at a rail location
        Railway_Loc railway_loc;    // railway location the train goes through for an arrival time
        Wait_time;                  // time to wait during an arrival at station  if it has a wait associated with it
}
```

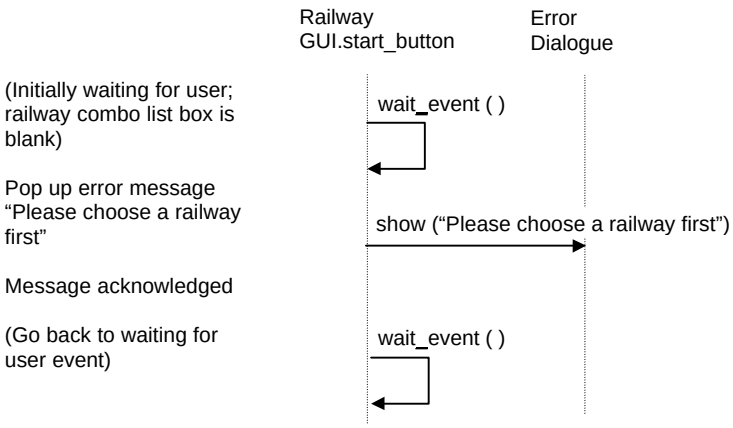
**Start button pressed: Railway time clock initialized to earliest train departure that is stored in all train objects; time compression initialized; train objects initialized to their starting locations**



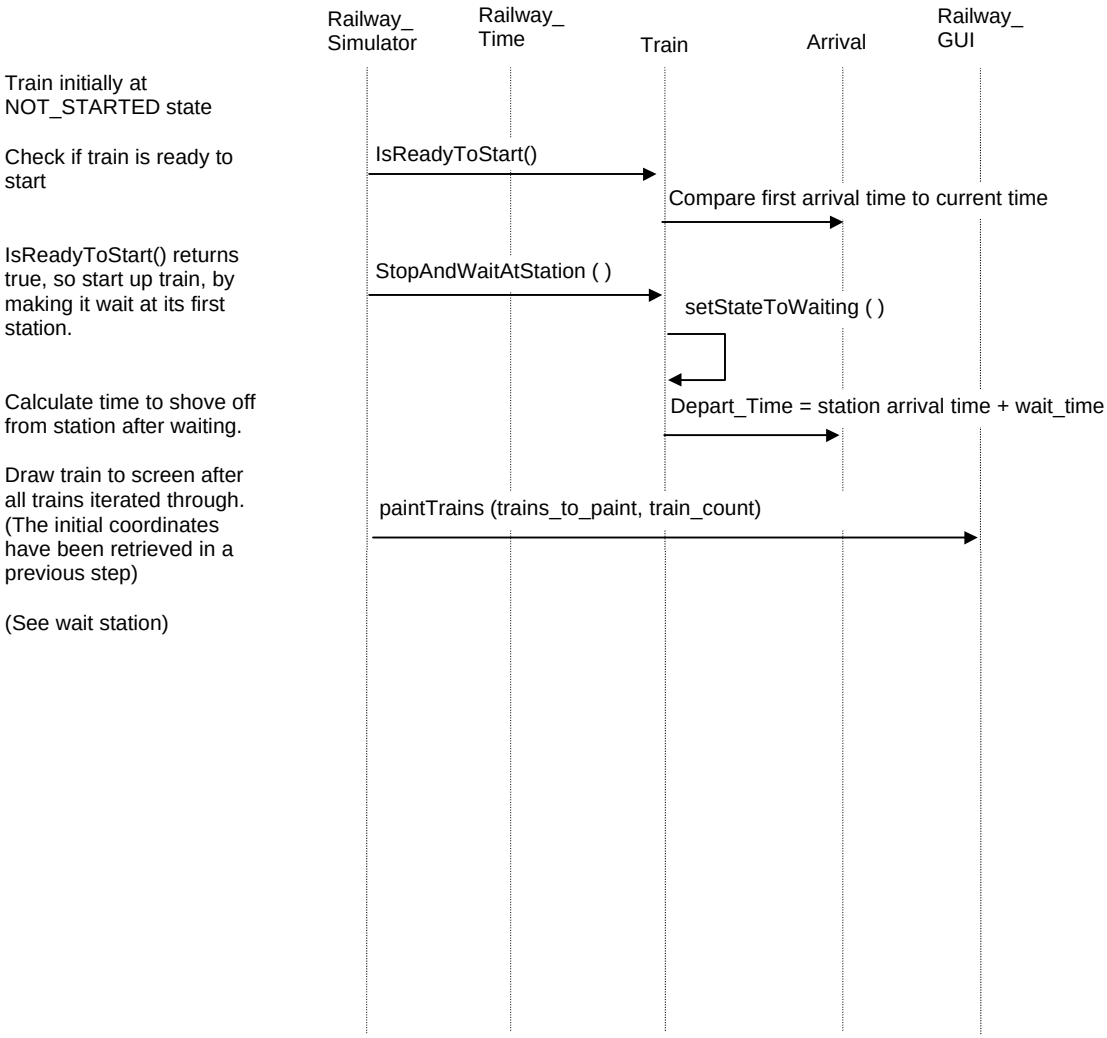
**Start button pressed: Railway simulation starts running and railway bitmap loaded simultaneously**



**No railway selected when Start button pressed**



**Time for train to start arrives and it is drawn to screen for the first time.** (All trains are iterated through in a for loop, which in turn is in a loop until simulation end detected.)



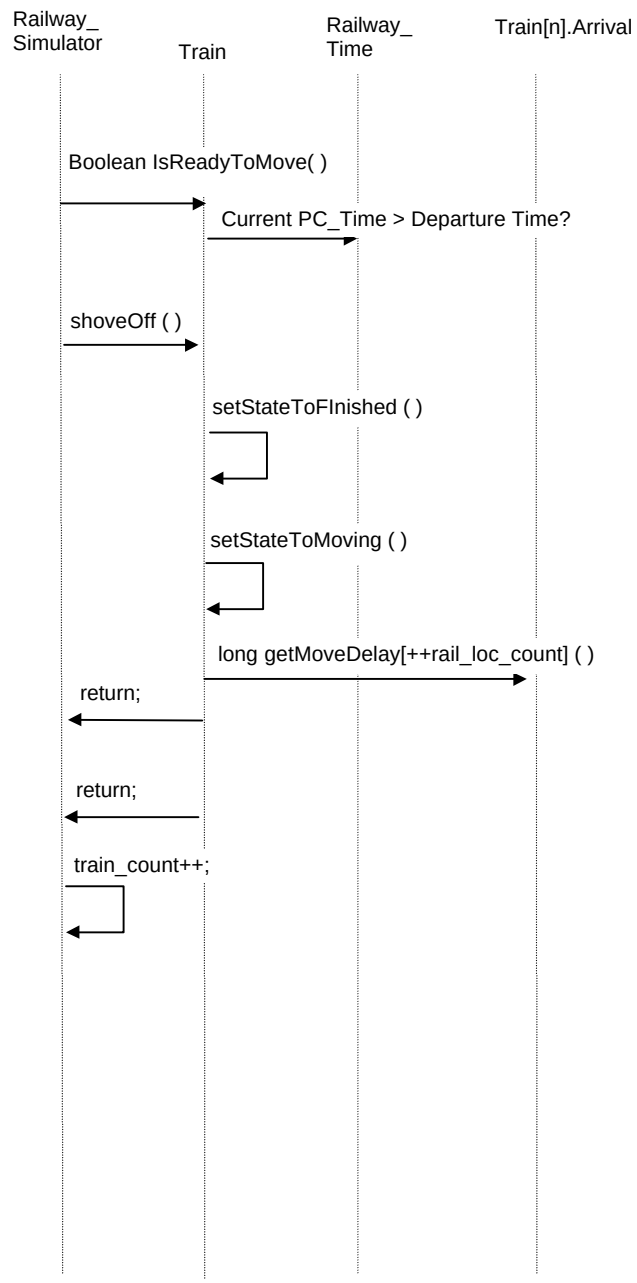
**Train objects running: Train is stopped at station and awaits departure to next station** (Includes train arriving at first station in line, and train object arriving at final station in line and it's state set to FINISHED.)

Train is currently in WAIT state, until departure to next station, or finish of its run

RailwaySimulator calls train to see if it's time for moving/finishing has occured

If current time >= departure time
  Train told to shove off from station
  If current station is the final station in line for train
    Set the state of train to IS_FINISHED and train will disappear from screen on next run
  Else
    set train state to moving and prepare midpoint algorithm to calculate movement to next station/angle coordinate.
    Get move_delay time for current segment
Else
  Railway simulator does nothing (train stays drawn at current station.

Railway simulator does operation on next train or draws all active trains



**Train objects are running: Train is moving from one station or angle coordinate to another station** (includes changing of waits when arriving at angle coordinate)

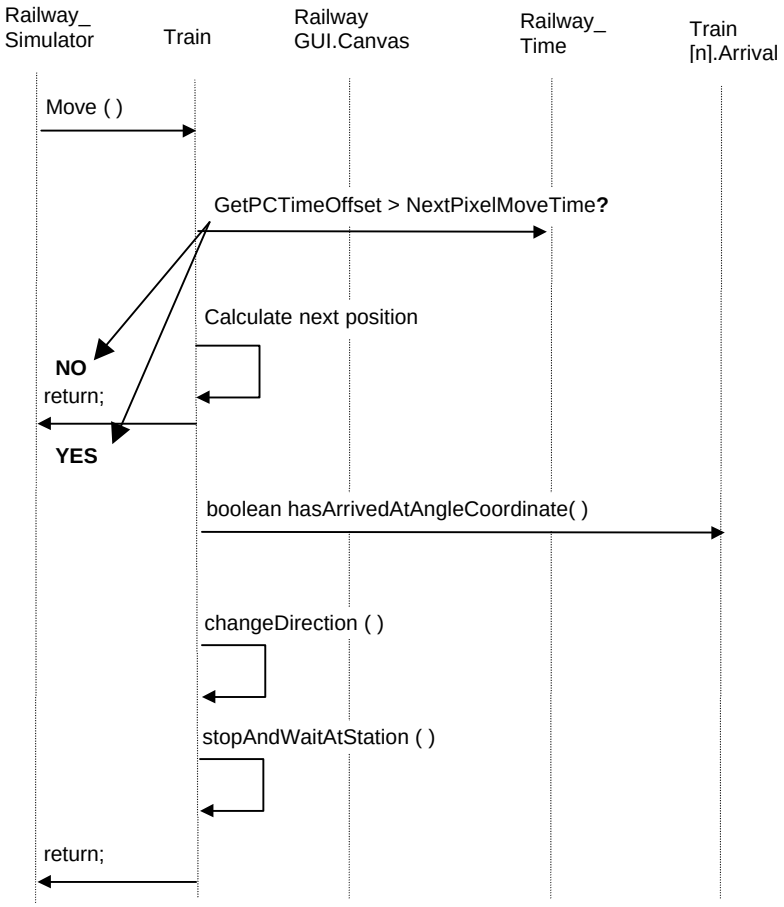
runRailway ( ) in Railway_Simulator tells train it can move

Train fetches current PC time to compare time to move with the time it's supposed to move to the next pixel

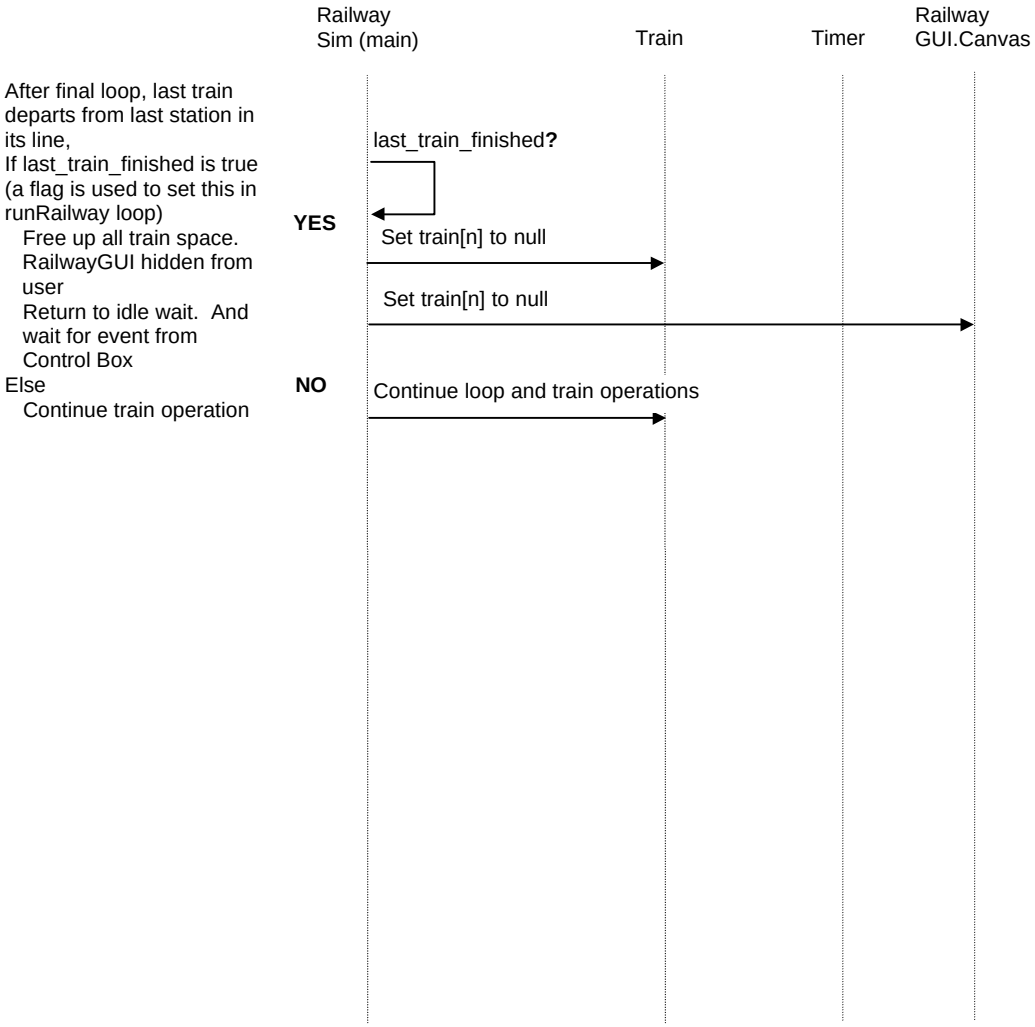
If GetPCTimeOffset >= NextPixelMoveTime
  Move the train to the next pixel using midpoint algorithm
else
  return to RailwaySimulator (no movement made)

If the train has reached the last pixel of the current segment movement:
  If location, the train is now on, the coordinate of the next angle coordinate
    Prepare midpt algorithm to calculate change of direction on the next time the train moves
  Else train has arrived at station so prepare train to wait at station

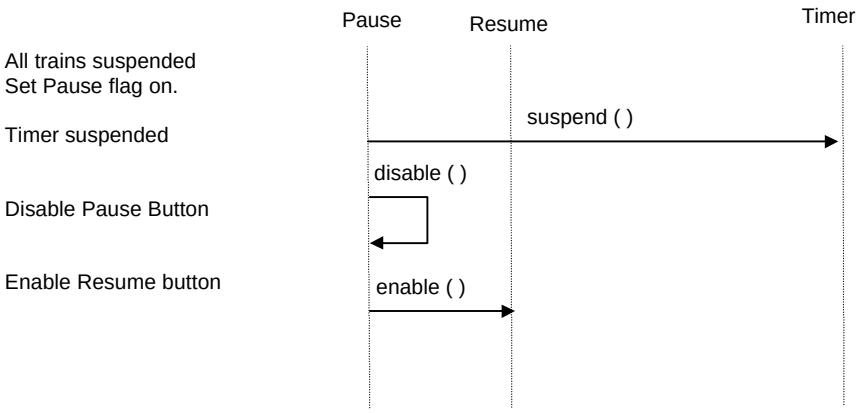
Return to calling railwaysimulator so current pixel location may be drawn



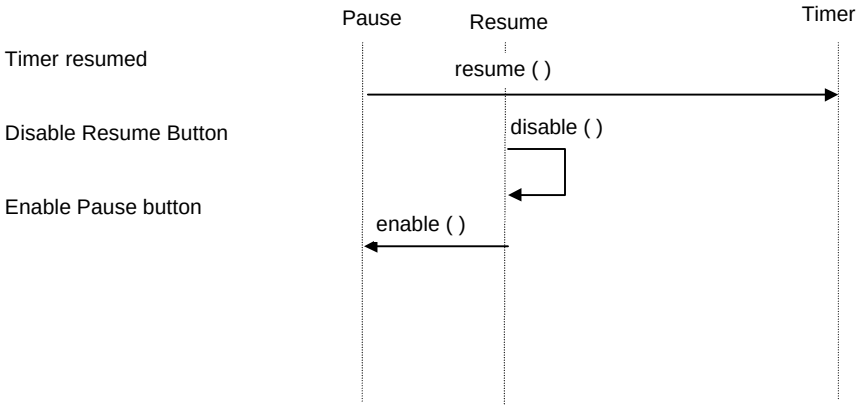
**Final train in entire schedule arrives at last station; simulation ends and simulation put in stopped mode automatically**



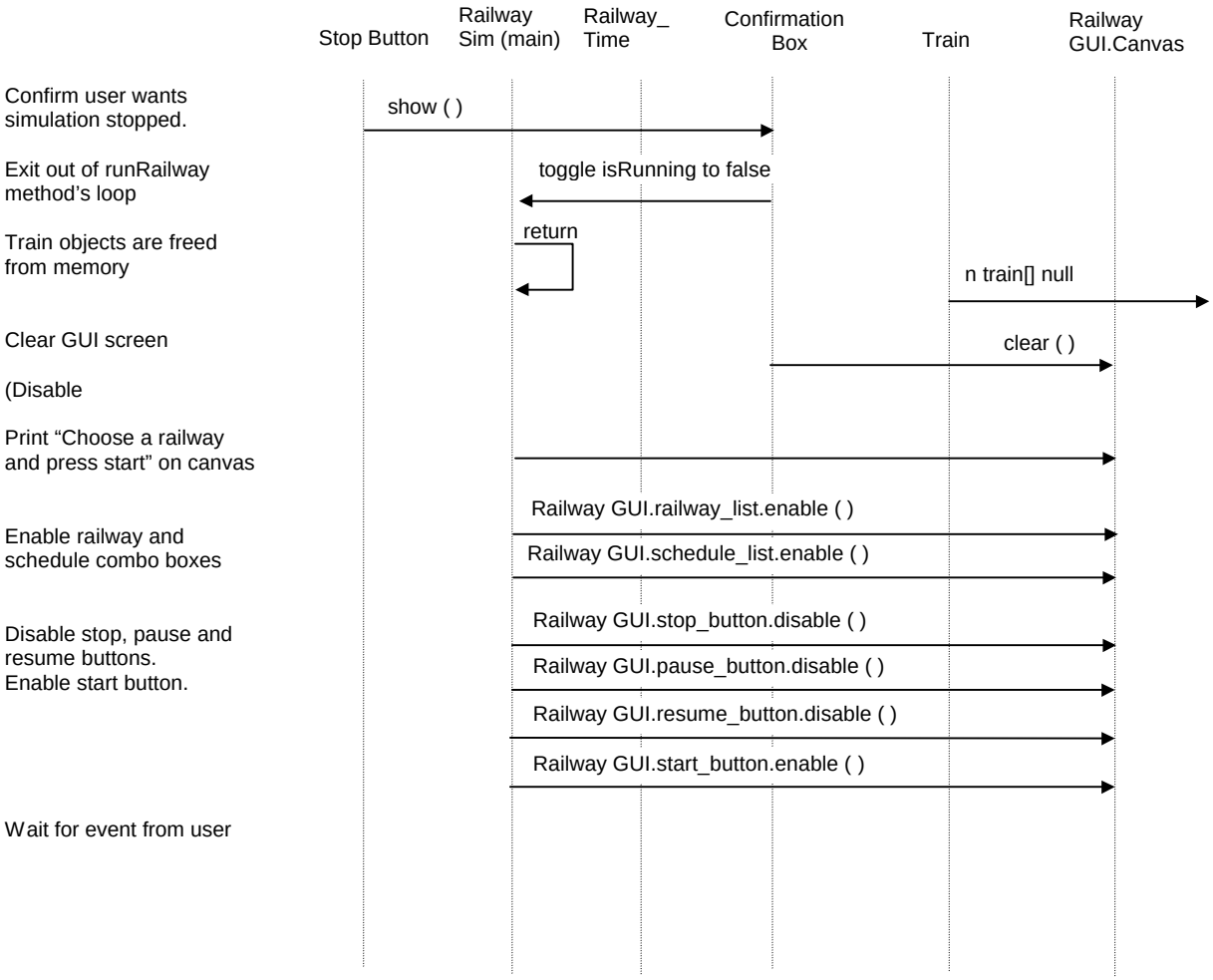
**User presses the Pause button** (in running mode)



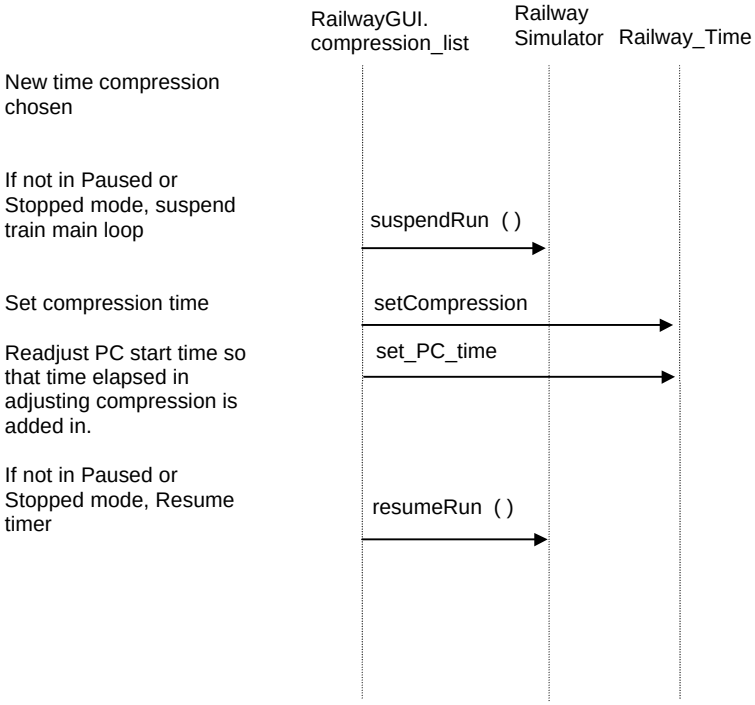
**User presses the Resume button** (in paused mode)



**User ends railway simulation by pushing stop button** (in running or paused mode)



**User chooses different time compression factor** (any mode)

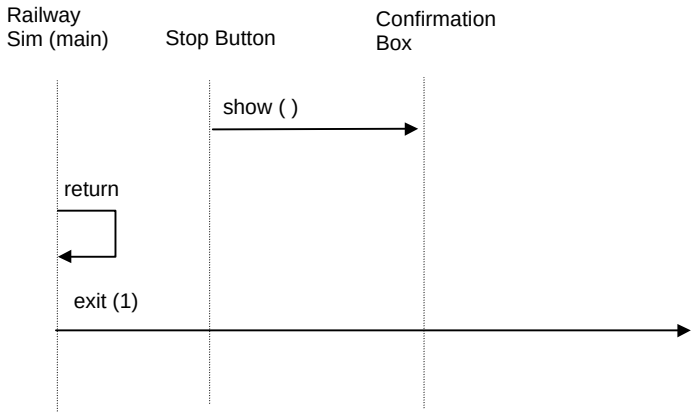


**User quits railway simulation application by pushing quit button** (any mode)

Confirm user wants simulation application ended.

If confirmed yes and not in Stopped mode, runRailway method is exited.

Railway Sim exits its context; all objects part of it are destroyed.

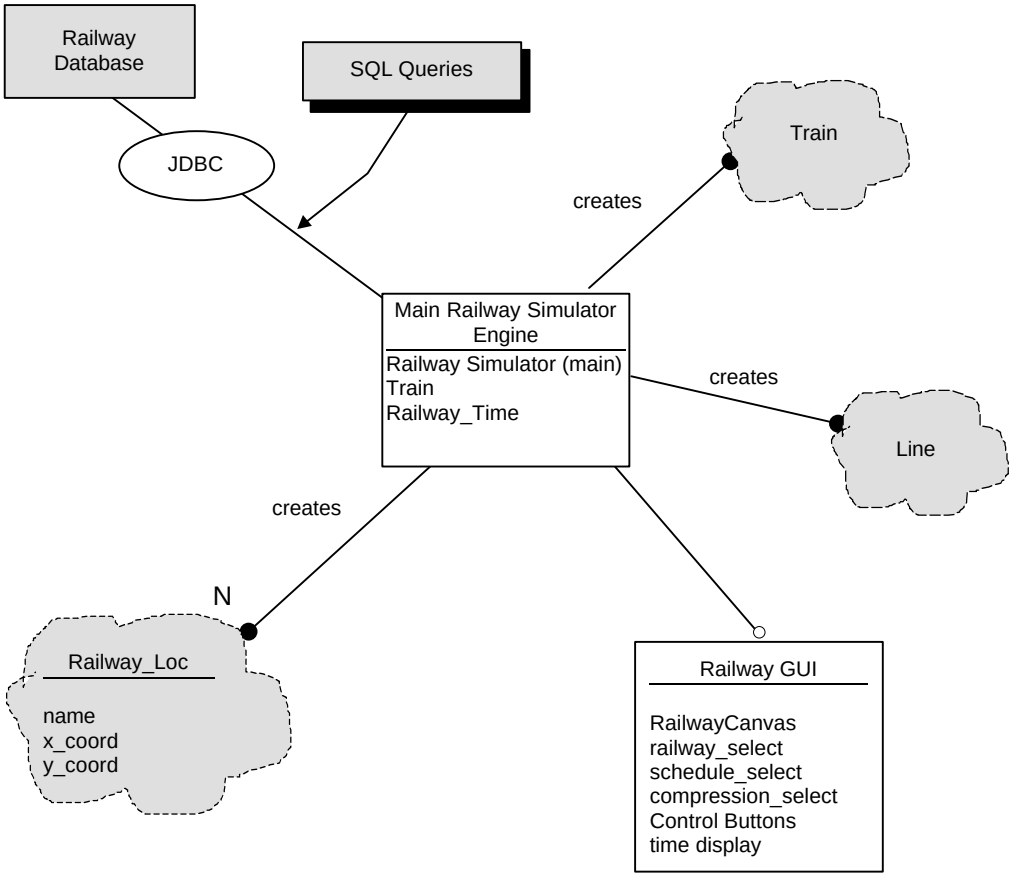


# SECTION 4

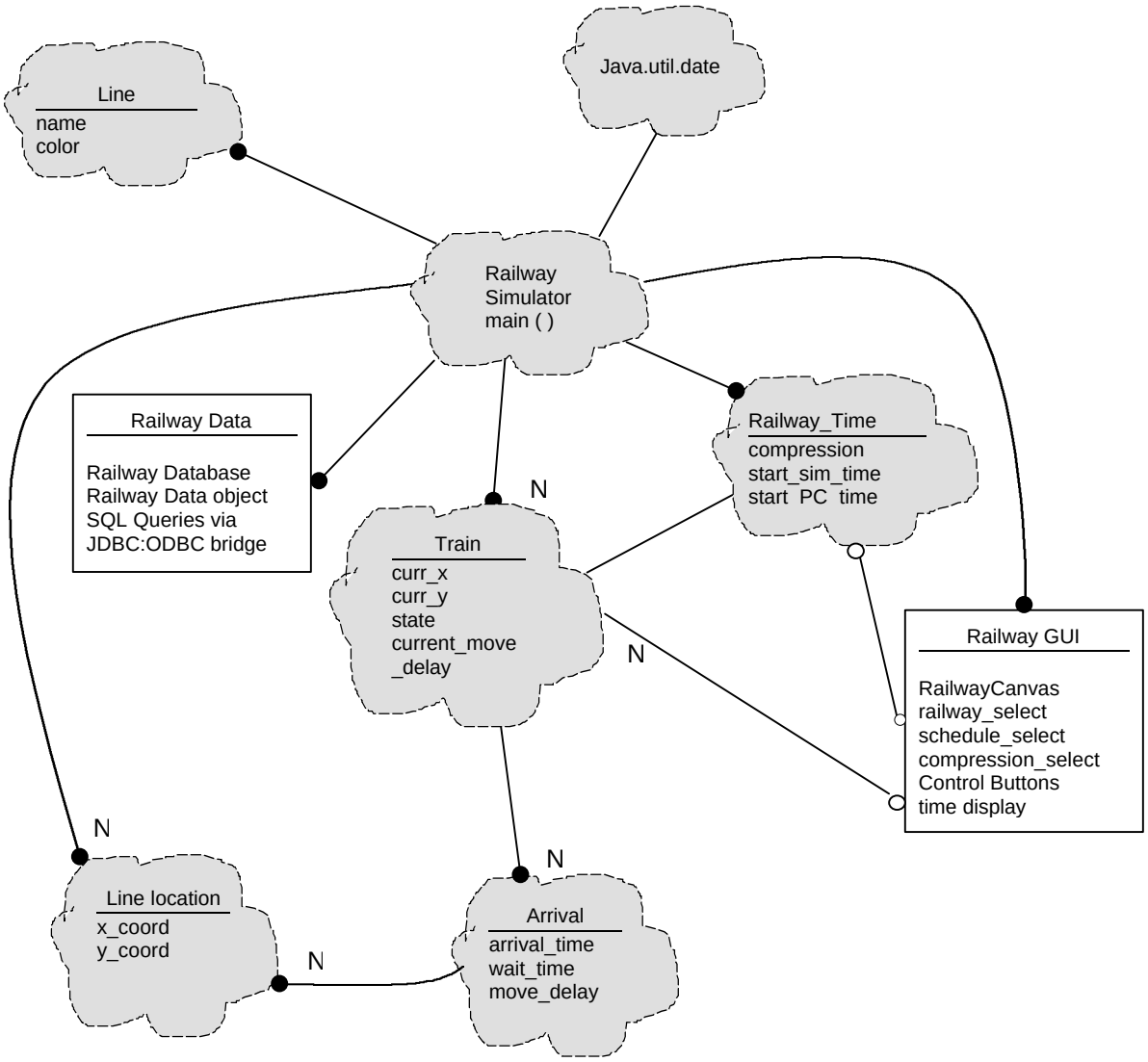
## Class Diagrams

This section displays the classes and relations identified for the Real-time Railway Simulation project in a diagram format.

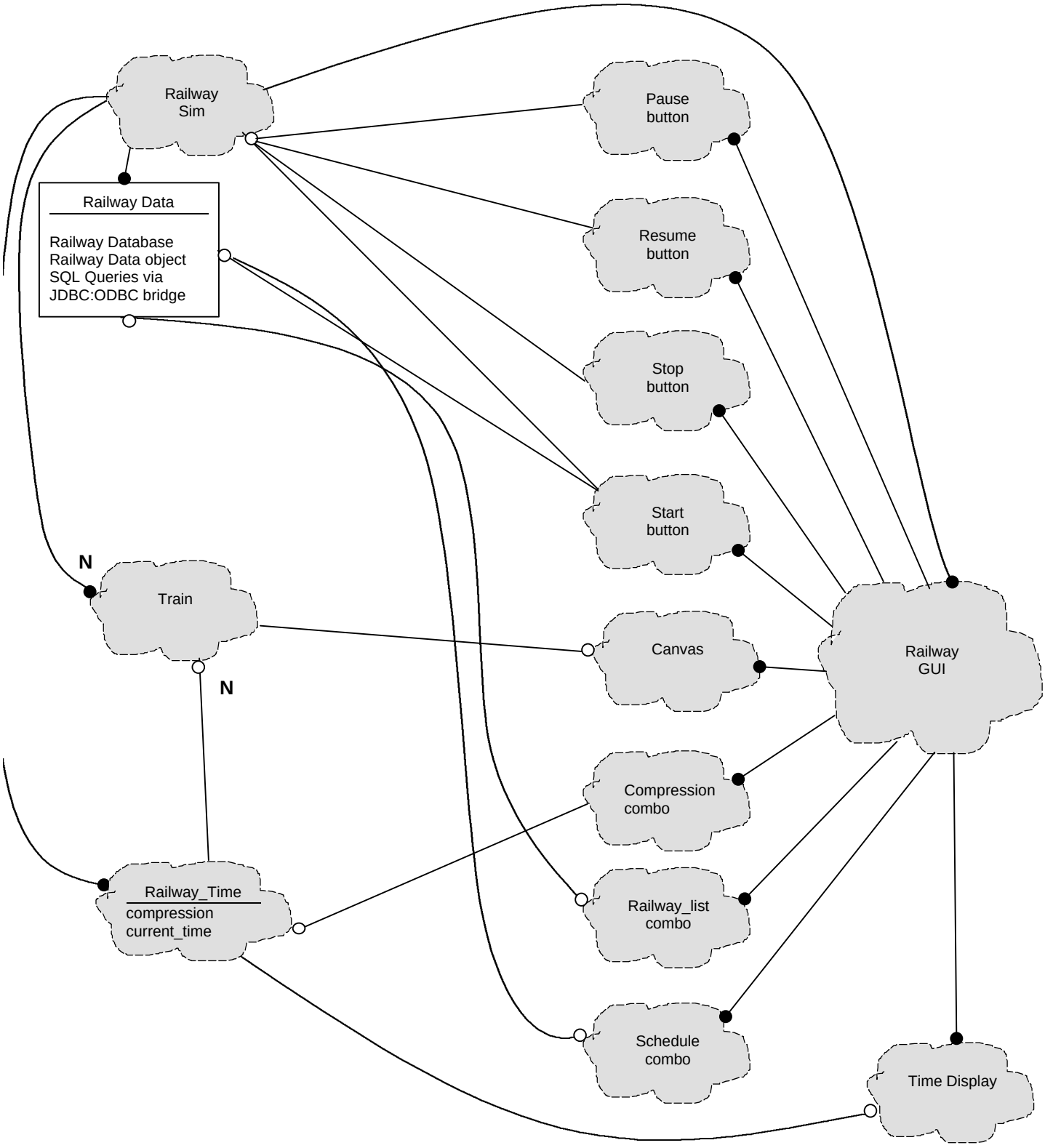
**Classes and class relationships with focus on Railway Data Area**



**Classes and class relationships with focus on Main Railway Engine area**



**Classes and class relationships with focus on Railway GUI Area**

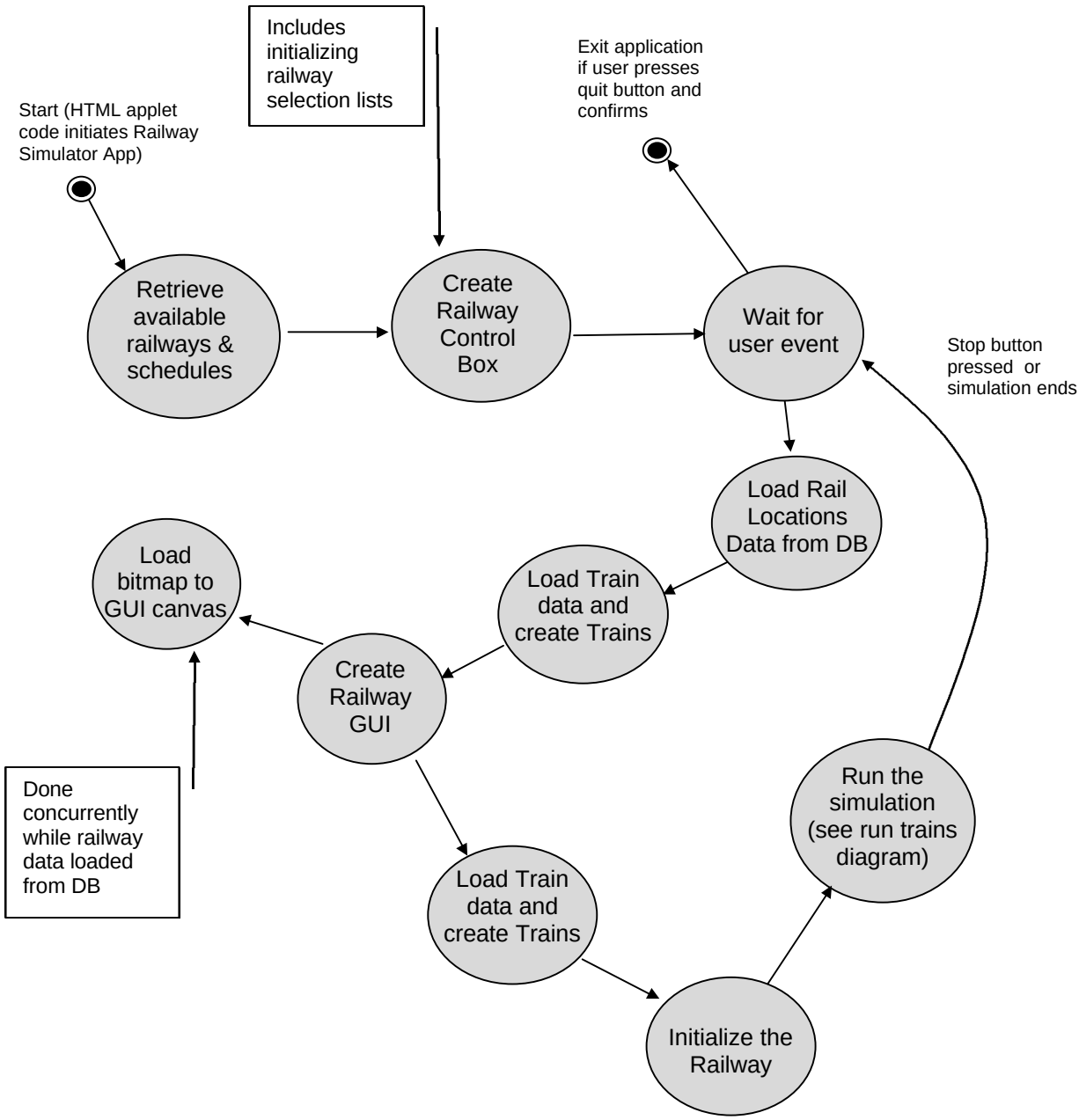


# SECTION 5

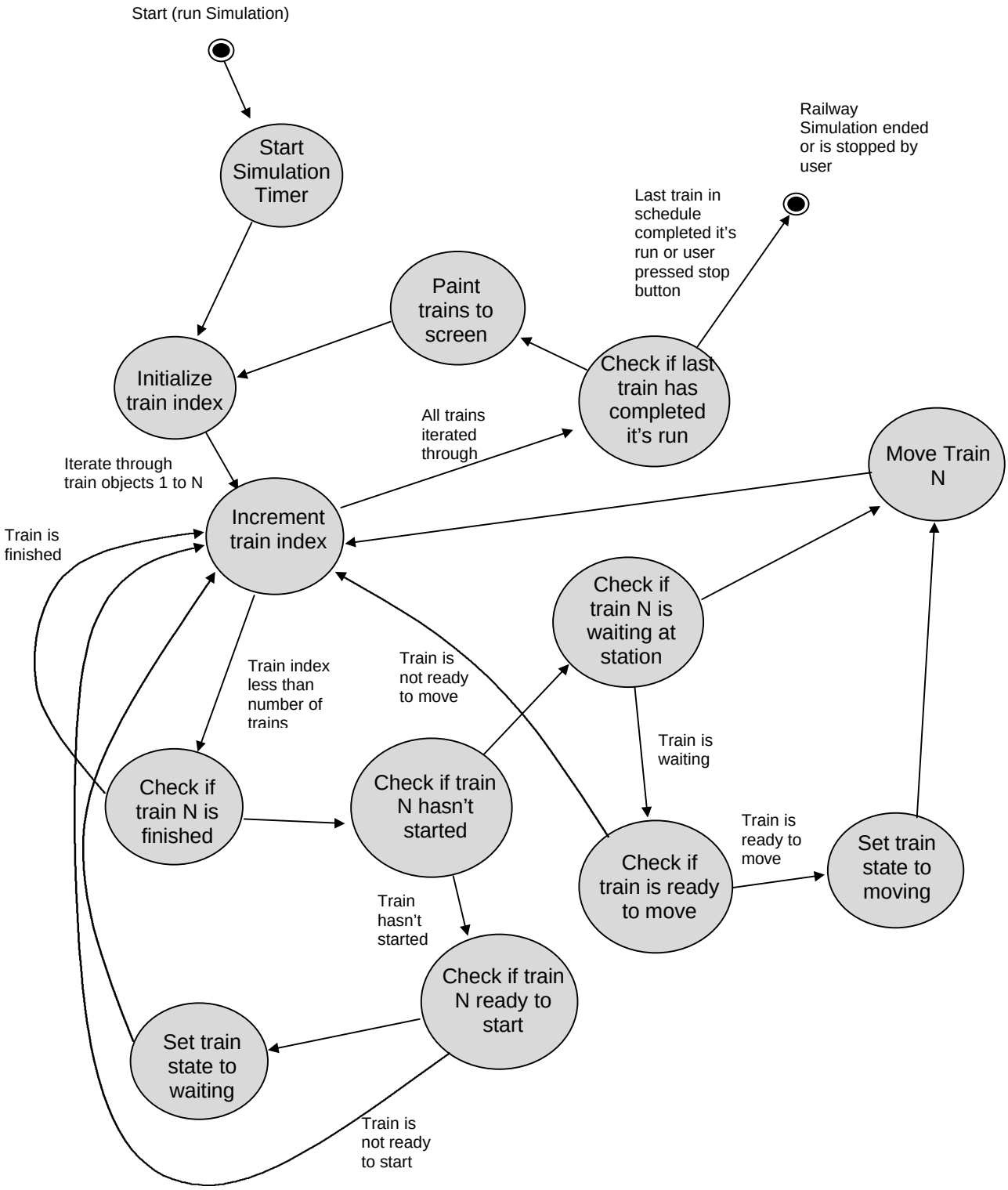
## Class State Transition Diagrams

This section displays the state transitions of applicable classes in the Real-time Railway Simulation project.

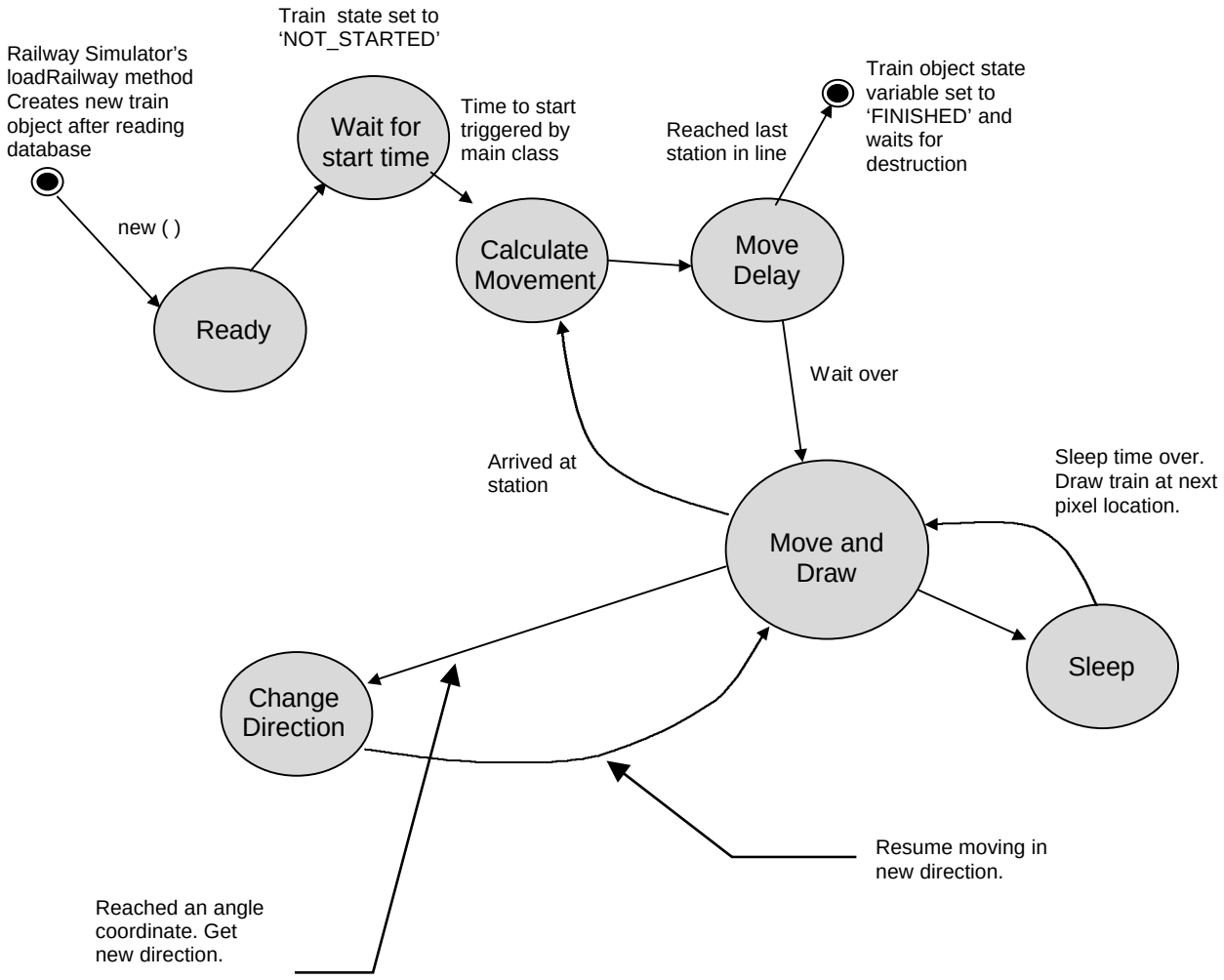
**Railway Simulation (main) Class State Diagram (load part)**



**Railway Simulation – run trains method – State Diagram**



**Train Class State Diagram**

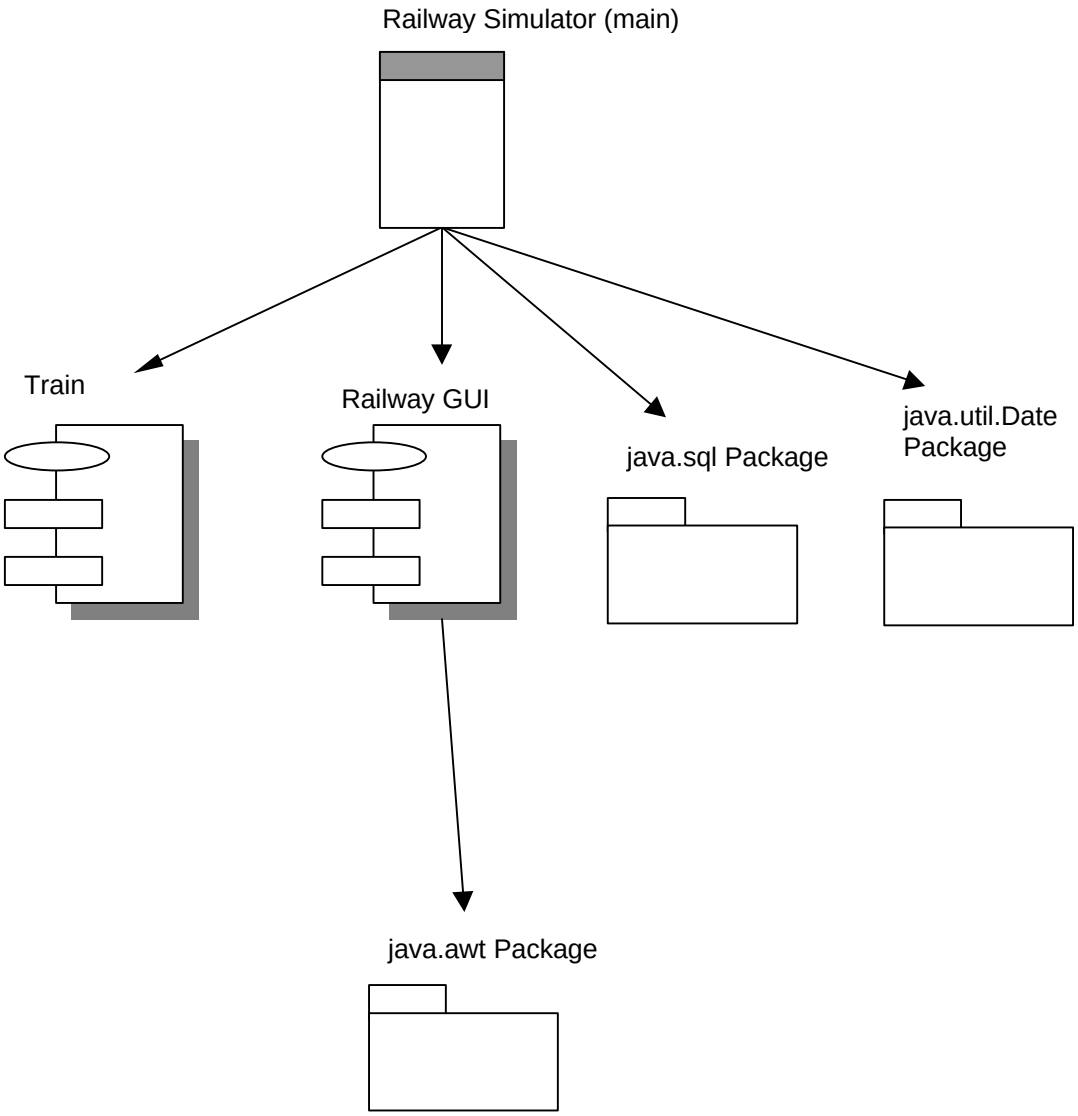



This diagram is closely linked with Railway Simulation – run trains section. It appears that the two are unconnected, but the railway simulation actually coordinates the train actions with the PC timer. The above view is inaccurate as a state diagram as the application actually operates, but gives a view of the order of states a train object goes through.

# SECTION 6

## Modules Diagram

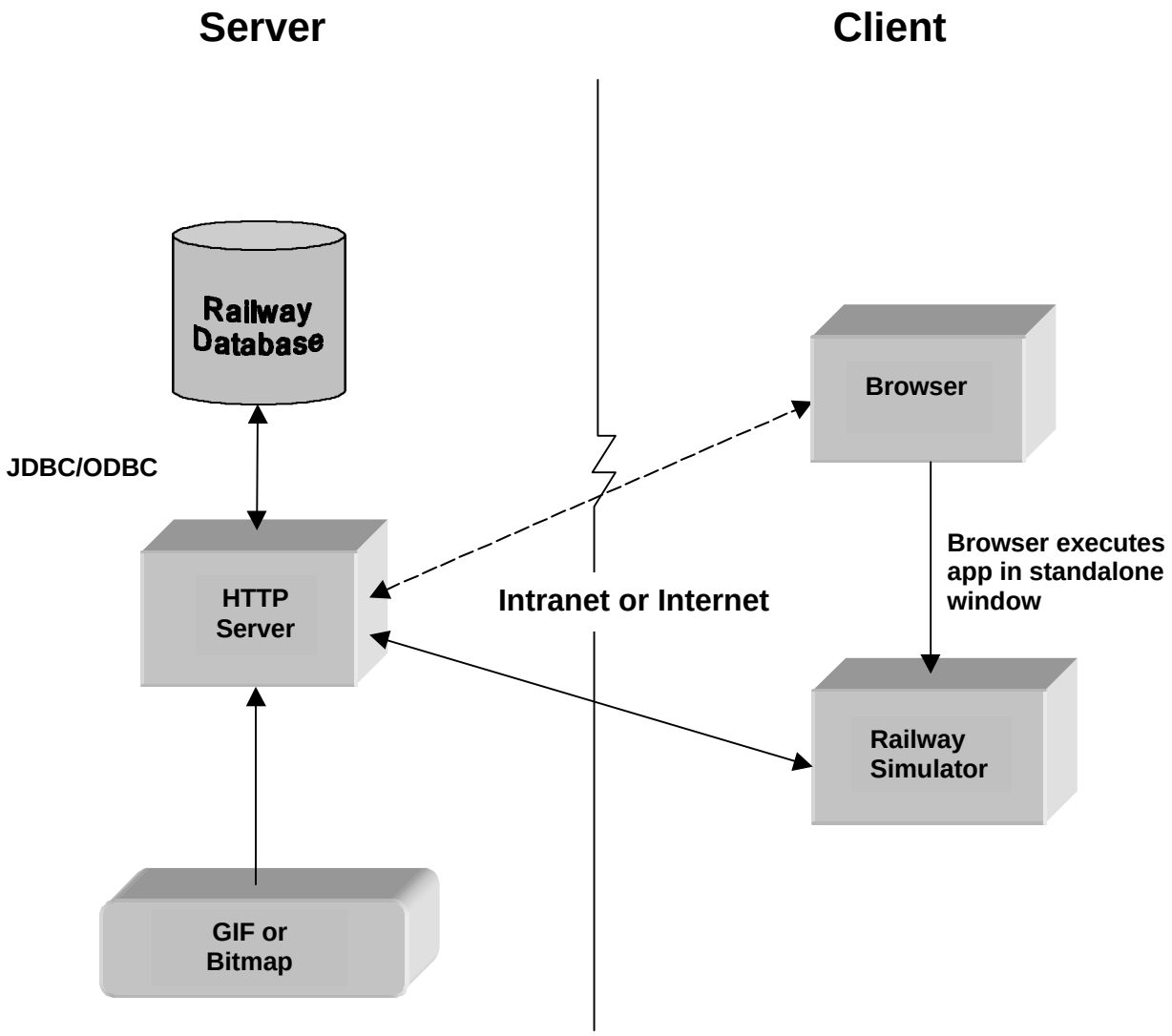
The following diagram is the proposed architecture for modules and Java packages used in the Real-time Railway Simulation.



# SECTION 7

## Process Diagram

The following diagram shows the proposed architecture of the distribution of the Real-time Railway Simulation project as it runs over the Internet or an intranet.



# SECTION 8

## Relational Database Design / Entity Relationship Diagram

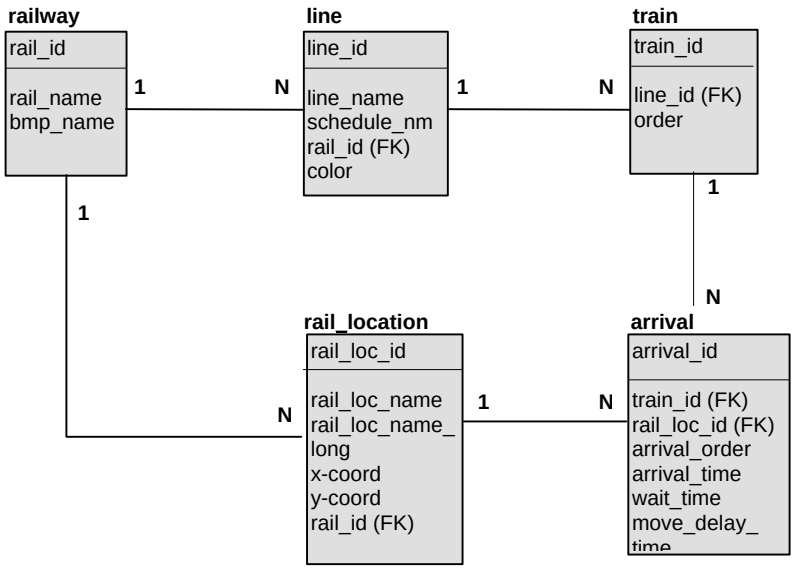



**Figure 8.1. Railway Database E-R Diagram**

**Business rules**

1) Each railway has a name and bitmap associated with it. The bitmap will have trains animated over it. This bitmap should be in the area where the application references it, which will probably be in the same server the application is loaded from.

2) Each railway has one to many line locations, which are either train stops or angle coordinates that the trains in each line travel through. (An angle coordinate is a point in the line where the train changes direction in the simulation area without stopping en route to the next station it is moving to.)

   Each line location has an x and y coordinate, and a time to wait. An angle coordinate will have a wait time of 0, since the train is merely changing direction in it, not stopping to pick up passengers as in a station.

3) Each railway has one to many lines. Each line is associated with a single schedule (e.g. Weekday, Saturday, Sunday, etc.). The line will have a color attribute, which will be the color painted for its trains running in the simulation.

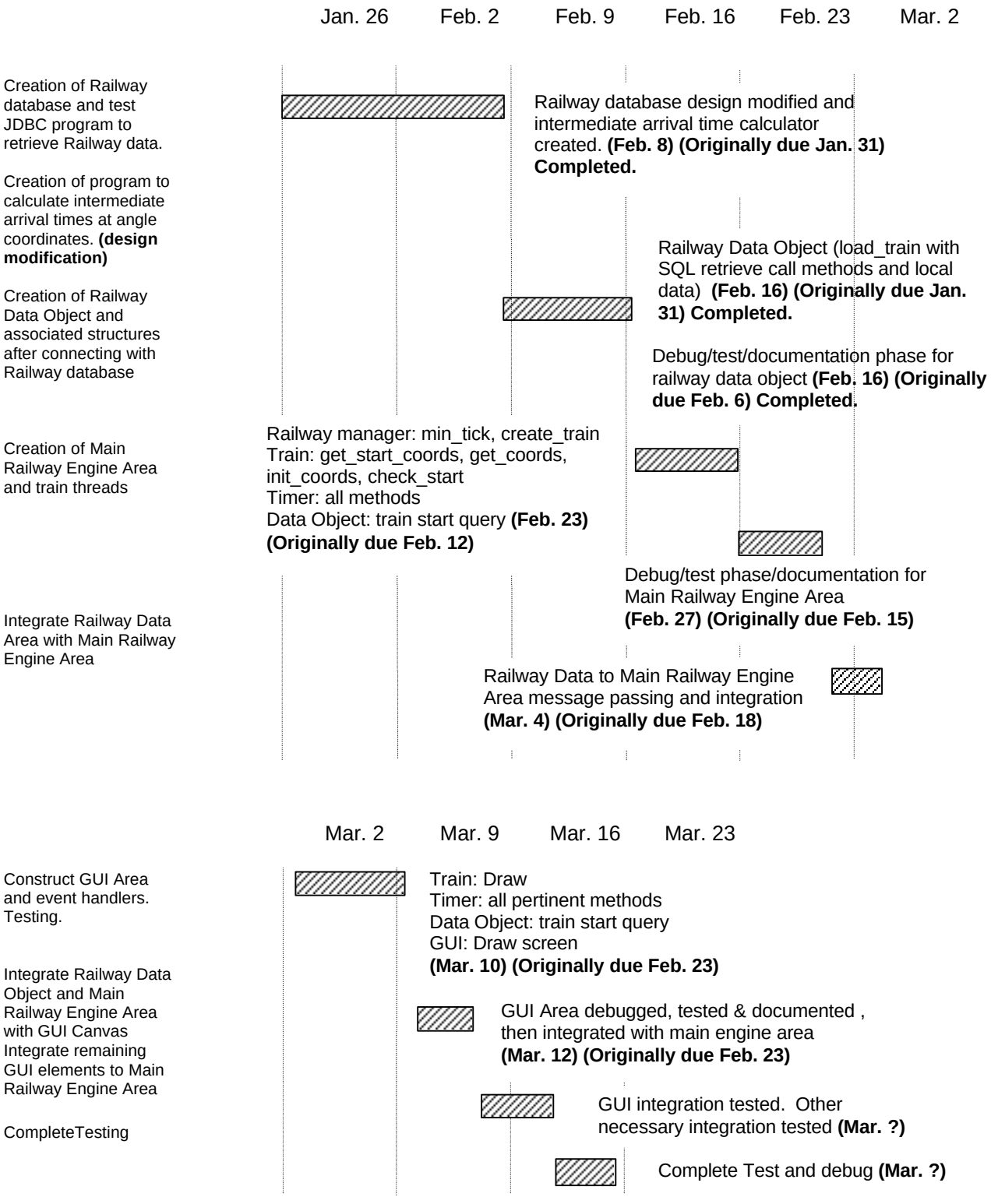
   Lines having common endpoints and station stops, but travelling in different directions are considered separate lines. (e.g. 'Daly City to Fremont-Weekday' and 'Fremont to Daly City-Weekday') are considered two distinct lines.

4) Each line has one to many trains, which run through it in a single day.

5) Each train has a set of arrival times for each of the line locations assigned to it. The station arrival times come directly from the schedule given by BART, Montreal Railway, New York Subway, etc. The angle coordinates that may be between stations are derived from these station arrival times and the coordinates of angle coordinates. Angle coordinates will have no wait time associated with them; trains just pass through it. While stations will have a wait time to simulate loading and unloading of passengers.

6) There is a many to many relationship between line locations and arrival times. Trains from different lines may share the same station(s) with each other.

# Implementation Plan in Winter Quarter, 1998



# Real-time Railway Simulation issues report with test case descriptions

**Jan. 25** - problem thinking up redesign of database
Trying to make automated program from text, but multiple arrival times in trains may need to be created by hand.

**Jan. 28** - Redesign of data structure to accomodate above issue

**Feb. 1** - Not able to create foreign keys in Access SQL call. FIXED

**Mar. 20** - completed v0.9 of Railway DB retrieval code
Test cases for reading of text file data into database:
1) Read of rail locations into rail loc array. PASSED
2) Angle coordinate times calculated correctly using Midpoint algorithm
3) All times are in correct and increasing order taking account of angle coordinates added to it.
4) All move delay times are calculated correctly and stored with each angle coordinate it belongs to.
5) All wait times are stored with each station. (Though currently all wait times are awkwardly assigned a static 30 seconds.)
6) Move delays for transitions between stations with no angle coordinates in between are stored in each station. Method: use JBuilder to step through each data piece

**Mar. 24**
1) Arrival.wait_time is being converted from float to int; specifically, a 0.5 in the database becomes 0 in local memory.
2) Make sure Arrival.arrival_time is being stored correctly. I can't see for sure in JBuilder IDE
3) There is an issue with modifying a Time object value. The solution, which I will stick to for now is to never modify the Time value. Only modify the string which is used for creating the time object. This was being doneto add a wait time at train stations.

**Mar. 27** - create railway simulation initialization routine
Problem: Making copy (clone) of another object. A workaround was made for this. This has to do with the train coordinates. Unresolved but minor issue.

**Mar. 28** - mapping simulation time to the PC's time
The method I propose to use to time the movement of trains is to synchronize against the PC's clock.
At initialization, the time of the earliest train departure is marked as the beginning time in the simulation. This will be represented by the Time object (which holds the number of time elapsed in milliseconds from midnight to this beginning time, plus 28800000 milliseconds for some reason).
This will be mapped against the PC time.
When the trains are initialized and ready to run, the current time of the PC is recorded as the starting time. This will be represented by a Date object (which holds number of milliseconds elapsed since January 1, 1970)
These will be subtracted from the current time in their corresponding contexts during simulation run, and the resulting offsets will be compared against each other for synchronization.
Some calculation will be needed when train running time crosses over midnight (00:00:00). I don't think assuming that a railway day always start in early morning and end late night or early morning the next day is a good idea. But due to project time constraints, it will be implemented this way for now.
I found that once the midnight hour is passed, it can be represented as 24:00 and times like 1am can be 25:00. This without rolling the internal millisecond value back to 0 (or rather 28800000). So the value held in the arrival_time, should be converted to this

**Apr. 1**

Check to see that the earliest departure time is being loaded from each trains' earliest arrival time (the first station it departs from). I had trouble getting the algorithm to work, but fixed.

**Apr. 3**
I added movement delay calculation to Arrival table to prevent calculation of it when simulation runs… a sacrifice of more space for more speed. However, I forgot to add in the the time past midnight during the calculation, and there are negative numbers on the intermediate location areas which have a starting station time before 12AM and ending station after 12AM.

My solution to above is as follows: The 'cutoff' time between the beginning of a railway day and the end of it is defined to be 3AM. That is the earliest train start will be assumed to be at or after 3AM and the latest train start will be assumed to be before 3AM. Maybe in a much later revision, this can be generalized. This is used to identify the start time of the railway simulation, in iterating through all the train first departure times.

**Apr. 7**
When should time be sampled from PC clock? Right before it's needed (before time offsets compared), or at a common time before the iteration through all trains for movement. If the latter, there may be a larger time difference as calculations go on, leading to greater lags for later trains in the Trains array but if the former, what disadvantages may come? I am tentatively using the first method (sampling the time right before it's needed)

**Apr. 8**
I consolidated of most time-related data into a single class named 'RailwayTime' since this seems to be a logical grouping.

**Apr. 9**
The problem with mapping of rail_loc_id in RailLocation object to an arrival reference to the rail_loc_id. Ideally, we want a rail_loc_id to be mapped as an array index in local memory, so searching for it is quick.
The problem is this mapping has to be done while loading because the rail_loc_id may not be starting from 0 to n. And the order of retrieval from the database may not be in numerical order. My solution is to do this mapping while initializing the simulation before running.

**Apr. 10**
I'm having a problem with how to handle situation where PC clock is going much faster (or much slower) than the computations when sampling time comes. I think a good solution would be to adjust the PC clock start time when a certain boundary of the expected comparison time is reached.
Is it really a good solution though? Ensuing problems?
Other options: 1) Automatically decrease or increase the time compression (like the original program does) 2) ?

**Apr. 11**
Eliminate checks for train arrival at rail locations if there has been no movement since last move delay. (This will eliminate unnecessary computation with a little overhead). !!This is not incorporated yet!!

**Apr. 12**
Need to add calculation of move delays if their are no angle coordinates between current station and next station. The 'InsertRailSim' program will do this.

**Apr. 14**
Issue with matching

**Test cases for loading of Railway Database data into local memory:**
(Compare these with the actual data in the database.)

1) Railway name and bitmap name received correctly. ***OK, but railway selection not implemented 4/27***
2) Railway Location object array loaded with coordinate data and names (and the array indexes mapped properly to the rail_id so these indices are used as new IDs which the trains may refer to). ***OK***
3) Line object array loaded with correct names and other data. (Lines are also mapped to an array index for easy reference by Train objects; this will be used mostly for referring to colors to paint the trains with). ***OK***
4) Train objects are mapped to the correct Line object array index as described in case 3 above. ***OK***
5) Train objects have the: ***ALL OK***
    a) correct number of arrivals
    b) correct order of arrival times
    c) arrival times match
    d) rail location index matched properly with corresponding rail location in the database
    e) move delay times are loaded correctly
    f) station wait times are loaded correctly

**Test cases for algorithm of running railway simulation:**
For each train:
1) All arrivals are being arrived at (do count through)
2) Correct coordinates are being arrived at and in proper order for each rail location (angle coordinates and stations).
3) Move delay times should be adding properly.
4) Wait at each time is equal or very close to its allotted wait time
5) Arrival time at each station matches that of the schedule (taking account of time compression)
6) Difference between start of waiting at a station and departure after waiting not too great or negative. (Here again is the issue of a slower or faster clock)
7) Trains are exiting out OK.
8) When implemented, check that PC start time adjustment, is OK.

USER'S GUIDE

# Real-time Railway Simulation in Java

Rev. 1.0

June 3, 1998

# Preface

This is the user's guide for the Real-time Railway Simulation in Java application. This guide will show the user how to run the simulation and also create custom railways (as well as other 2-D simulations). As a senior thesis design project, it is not a description of the complete desired project, but of what was completed before the end of the 1997-1998 school term.

There are two parts to this manual. The first part is a guide to running the simulation and using the graphical user interface. The second part is a guide to creating and inserting data so users may create their own custom railway. Then a short wish list of the desired complete product follows.

Following are some conventions used in this manual:

**Bold** words denote emphasis**.**
Words in i*talic* denote places elsewhere in the manual to refer to.
Words in `courier` are text examples for users to follow in designing their own custom railway.

# Introduction

Welcome to the Real-time Railway Simulation in Java. This is a recreational application used to 'view' a two dimensional animation of a railway system. Multiple trains on multiple railway lines can be simulated as a whole day elapses.

The simulation can be customized to run any railway or any two-dimensional simulation if you're willing to enter data in schedule format like a railway. The user has to provide the schedule information, and coordinate information in text format, and a bitmap over which the trains will be animated. The format will be described in this manual.

The simulation was originally designed to run over the Internet, by clicking on a link on a World-Wide Web browser, but is not currently implemented as of the time of this user-manual release. It is implemented as a stand-alone command-line driven program running on a Windows 95 or Windows NT machine.

Following are the requirements to run this application as it is currently implemented.

**Hardware requirements**
Intel Pentium computer with clock speed of at least 133 MHz.

**Software requirements**
Operating System: Windows NT 4.0 or Windows 95
Java Development Kit (JDK), version 1.1.5 or higher for data insertion and running of simulation
MS Access 97 Relational Database Management System (soon there will be Oracle support and other)
Microsoft ODBC

After making sure the above requirements are in your computer, do the following:

**Java Development Kit Setup**
After installing the JDK on your computer, make sure the directory for the java binaries is part of the Windows path.

In Windows 95, this would be done by modifying the autoexec.bat file by appending the directory to the 'PATH=' line. For example,
`PATH= c:\jdk1.1.5\bin`

In Windows NT 4.0, this would be done by first, clicking on the Start, then pointing at Settings, then clicking on Control Panel. The Control Panel window will appear. Double-click the System icon, which opens the System Window. Then click the Environment tab, which will show your environment settings. Click the PATH variable in the text window of the 'User variables for…' section. If the JDK binaries directories is not included on your PATH string, click the Value text box and go to the end of the string if any, then append the Java binary directory string. For example, if your Java binary directory is c:\jdk1.1.5\bin, append the following to the end of any existing PATH variable:
`;c:\jdk1.1.5\bin`,
then click the Set button, or

if PATH doesn't exist in the 'User variables for …' section. Click in the variable text box, delete any existing text and type PATH. Then click in the value text field and type:
`%PATH%;c:\jdk1.1.5\bin`,
then click the Set button.

**Installing the Java Executables**

Unzip the contents of the Real-time Railway Simulation into a single directory. For example,
`Pkunzip RailwaySim.zip c:\RailwaySim`,
or use Winzip to do the equivalent. The important executables necessary here, disregarding the source code files, are:

AngleTime.class
Arrival.class
Coord.class
CreateRailway.class
Inserter.class
InsertLine.class
InsertRailLocs.class
Line.class
RailLocation.class
RailLocInserter.class
RailwayCanvas.class
RailwayGUI.class
RailwaySimulator.class
RailwayTime.class
StationInfo.class
Train.class

If one or more of the above are not present in the directory you unzipped them to, type the following to make new Java executables:

```
javac RailwaySimulator.java
```

Disregard any of the warnings that may appear while compiling. (These will be fixed in a later update.)

Also, the following sample BART data files should be present:

Railway_Locs.txt
colma1.sch
colma2.sch
dalyci1.sch
dalyci2.sch
dublin1.sch
fremont1.sch
fremont2.sch
pittsb1.sch
richmo1.sch
richmo2.sch

**ODBC Setup**
There should be an ODBC connection setup first before this Java application can communicate with the MS Access database using Sun's JDBC/ODBC driver.

To set this up, do the following:
Click the Start button. Then point at Settings and click Control Panel. In the Control Panel window which opens, click the ODBC icon.

An ODBC Data Source window opens. Click the User DSN tab, then click the Add button.

In the Create New Data Source window that appears, click the Microsoft Access Driver list item in the text box, then click Finish. An ODBC Microsoft Access Setup window appears. In the Data Source Name text field type the following string exactly: 'Railway Database'. Optionally, a description may be typed in the Description field.

Click the Select button. In the Select Database dialog box that opens, you must point the data source at the Railway Database.mdb file. This should be in the same directory where you unzipped the Railway Simulation files. So either, by typing the whole path and file name in the Database Name text field, or by clicking on the folders in the Directories text field, select the database called Railway Database.mdb in the directory you unzipped the files to.

Once this is done, click the OK button. In the ODBC Microsoft Access Setup, click OK, and this should create the new ODBC data source called Railway Database. To confirm this, 'Railway Database' should appear in the User Data Sources text box.

Click the OK button to finish.

**There is no Railway Database.mdb file!**
If by chance the Railway Database.mdb file does not exist in the railway directory, a new blank Railway Database.mdb file must be created.

There are two ways to do this:

1) Start up Microsoft Access. In the small dialog box that appears in front of the larger MS Access window, click the Blank Database radio button, then click OK. In the dialog box that opens, click in the folder pull down menu to point in the directory where you unzipped the Railway Simulation files. After this is done, click in the File Name text field, delete any existing text, then type 'Railway Database'. Then in the Save As Type list box, make sure Microsoft Access Databases appear. Then click the Create button, to create the Railway Database.mdb file

**OR**

2) Click the File menu, then click the New Database menu item. In the New window, click the General tab. Double-click the Blank Database icon. Using the directory folder icons, point at the directory where you unzipped the Railway Simulation files. After this is done, click in the File Name text field, delete any existing text, then type 'Railway Database'. Then in the Save As Type list box, make sure Microsoft Access Databases appear. Then click the Create button, to create the Railway Database.mdb file.

Do nothing else to the database afterward. Just exit the application by clicking on the 'X' at the upper right hand corner of the MS Access window.

# Custom Railway Data Creation

The railway that comes with this simulation is the San Francisco BART system. More railways (or any kind of 2-D simulations) can be created if the data, which is originally stored in text files, and a bitmap file is provided. The data in the text files will be inserted into the databases. This procedure is explained in the next section. The creation of text and bitmap data is explained in this section. (As a note: it would be good to have existing schedules in disk storage already, as it will take a long time to enter all this time data.)

**Background**

Before instructing on how to create data for a custom simulation, a little background on how the application runs needs explaining. The railway simulation runs for one railway day. The first train in a railway day could start at, say 4:00 AM, and the last train may may end its run at 1:15AM the next morning, with multiple trains that run between those times. (A railway day is created and defined in the *Schedule Information* section, which is described later.) Throughout the day, multiple trains will run on multiple lines each train is assigned to run on. A line starts at one destination, for example, Fremont, and ends at another destination, say Daly City. Each train has designated stations to stop and wait at to simulate loading and unloading of passengers. Also, there are points where a train will not stop, but merely change direction; these are called 'angle coordinates' from this point on. The application allows only line movements from one point to another, whether angle coordinates or train stations.

With this definition of terms in mind, you are now ready to start creating data for the railway simulation.

**Bitmap**

Provide or create a map over which the trains will be animated. Make sure it is large enough and appealing to users. The acceptable bitmap formats are: JPEG and GIF. Look over the map and think how you want the trains to move for each line. Look at the stations and see how you want the train to move from one station the next, location-wise. Do this while creating coordinate information and schedule information in the next sections.

```
BART
bart.gif

RICH|160|100
RICH_TO_DEL-N_1|181|102
DEL-N|186|109
PLAZA|190|126
N-BRK|198|145
BRK|210|156
ASHBY|211|169
ASHBY_TO_MACAR_1|211|180
MACAR|214|191
19ST|215|201
12ST|212|207
LAKEM_INTERSECT|206|218
W-OAK|186|210
 ...
```

**Figure 1.1 Example of text file data for railway locations**

**Coordinate Information**
Using the bitmap, note down on paper or elsewhere, the X and Y coordinate of each station and angle coordinate. Then, give a short name for each station no more than 16 characters long. These same names will be used in the data file for storing the schedules of each line.

These will be put in a text file. This file can be named anything, but make it a meaningful name. A suggestion is to append the suffix '.loc' to the end of the file. Eventually this data will be inserted into the railway database.

A railway location text file is shown in Figure 1.1. This format must be followed if the application is to run correctly. The first two lines are general railway information. The first line contains the name of the whole railway. The second contains the name of the bitmap file used to paint the trains over.

One or more blank lines can be added between the second line and the next lines of data.

The remaining lines are the railway location data. As shown, each line is pipe delimited and contains 1) the name given to the station/change of direction location, 2) x-coordinate of the location, and 3) y-coordinate of the location.

Save the file to the Railway directory.

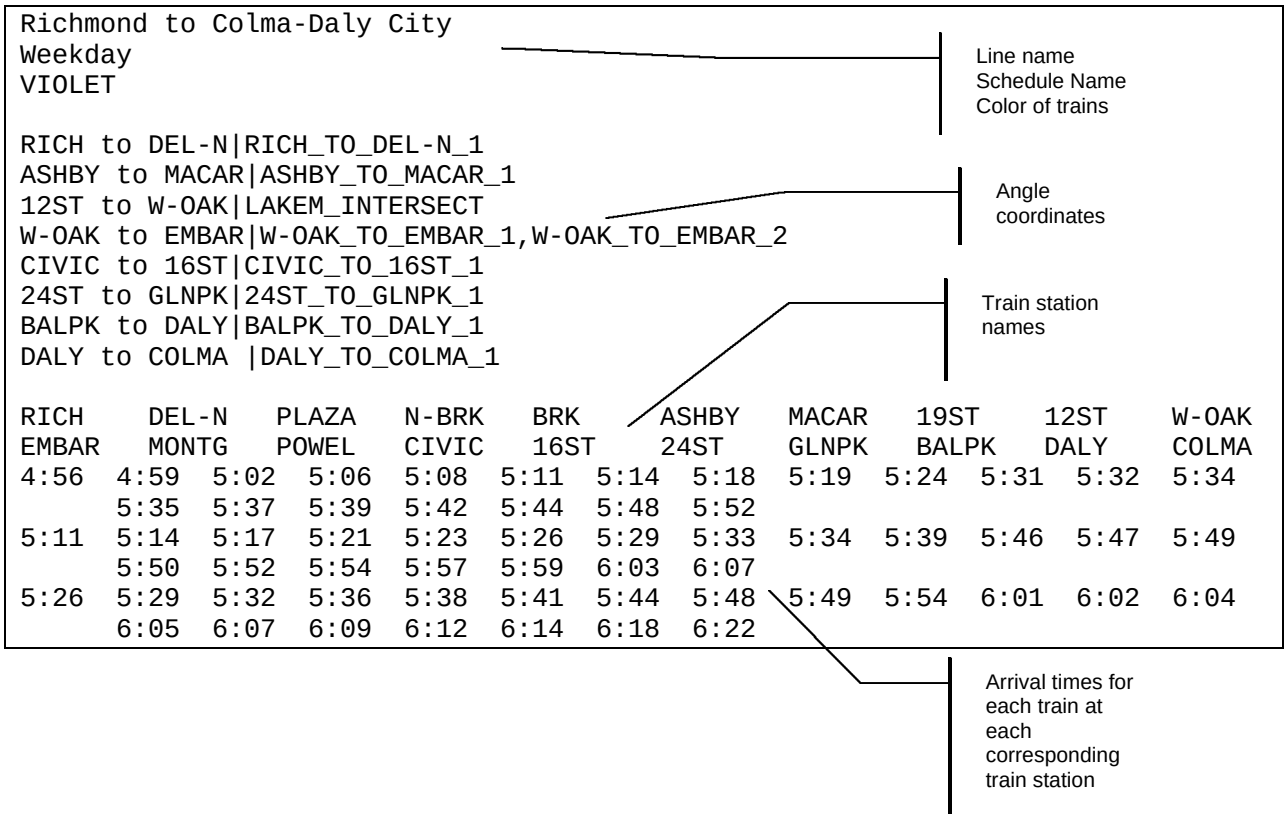



**Figure 1.2 Example of text file data for railway schedule**

### Schedule Information

Entering data for the schedule requires creating text files, just as the rail locations data file was created. A sample schedule file is shown in Figure 1.2, whose format must be strictly adhered to. Looking at the top two lines, note that the text file only covers schedule information for a single schedule in a single line in the railway. So, in order to have a railway with multiple lines each having or not having multiple schedules, a text file for each must be created.

Back to the schedule text file, the line name and schedule name must be the first two lines of the file. Following that, the color of the trains in this line/schedule must be entered. (The painting of trains in this color has not been implemented yet, but put the color in anyway.)

Following these three lines, are a series of text lines describing the angle coordinates used by this particular line. Note that a single pipe character delimits each line. Before the pipe, is a clause stating the two stations trains will move between. After the pipe, is the name of the angle coordinates a train will go through in this station to station movement. The names of these stations and angle coordinates must appear in the rail location text file word for word, including the case of the letters. Commas should separate each of the angle coordinate names following the first pipe. They must be in the order in which the train will move from the first station in the clause to the second station in the clause, unless you like goofy runaway trains.

To clarify, by looking at a real-world view of what's being defined here, each train will move from one station to another. A train moving from one station to another in just one straight line is boring (and unrealistic). So you'd want the trains to move through one or more points where it changes direction without stopping, before arriving at the next station. Going back to the line definition, the clause before

the pipe states the stations you want trains to move between, and the series of angle coordinates after the pipe states the name and order of points you want trains to go through.

The next singular line is the names of all the stations used by trains in this line. Note that each station name should be delimited by one or more spaces. These names must appear in the rail location text file, letter for letter, as well. Also every station mentioned in the angle coordinate clause must appear here as well.

The next several rows of lines are the schedule arrival times for each train in the line. Note that each line of text is the arrival times of a single train at each station typed in the station name text line, which precedes these arrival times. Each arrival time must match one of the stations (disregard the angle coordinates), and the times must be in sequential order left to right. Also, each train's arrival times must be in one continuous line. No carriage return breaks are allowed. Each time is in military time format, for example 15:00 represents 3PM, and each time is delimited by one or more spaces.

These schedule times will be the longest and most tedious part of the data entry. It is a good idea to have a schedule already made, such as train schedules from the Internet. The schedules that are included with the Railway Simulation application, were taken from the BART website, and slightly modified using the PERL programming language.

So multiple files for each line/schedule must be created. The text files can be named anything, but give it a meaningful name. Ending the file with a suffix of '.sch' or '.sched' is one recommendation. These files should be saved in the directory with all the other Railway Simulation files.

## Inserting text data into the Railway Database

After creating all the text files and checking that the data is correct, their contents can be inserted into the Railway database.

In the directory where the Java executable files are and where the previously created text data files should be placed, do the following to insert the data into the database:

1) Create a new blank Railway Database by typing and entering:
```
java CreateRailway
```

2) Then, insert line location coordinates into Railway Database by typing and entering :
```
java InsertLineLocs [railway location text file]
```

3) Then for each line and associated schedule, type and enter:
```
java InsertLine [line/schedule text file name]
```

Step 3 will take a while to insert into the railway database, perhaps up to 3 minutes per file.

If the programs in step 2 or 3 error out, make sure that the data you put in the text files are correct. Then start over from step 1 again.

**Inserting data using the included BART Weekday Schedule**
Included with the Railway Simulation files are the data files for the BART Weekday Schedule. You may run all or some of the lines (it will take a long time to load).

To load the BART Weekday Schedule data:
1) Create a new blank Railway Database by typing and entering:
```
java CreateRailway
```

2) Then, insert the BART line location coordinates into the database by typing and entering :
```
java InsertLineLocs Railway_locs.txt
```

3) In the introduction section is a list of all the line/schedule files used by the BART Weekday simulation. For example to insert the dublin1.sch schedule file, type:
```
java InsertLine dublin1.sch
```

## Running the simulation

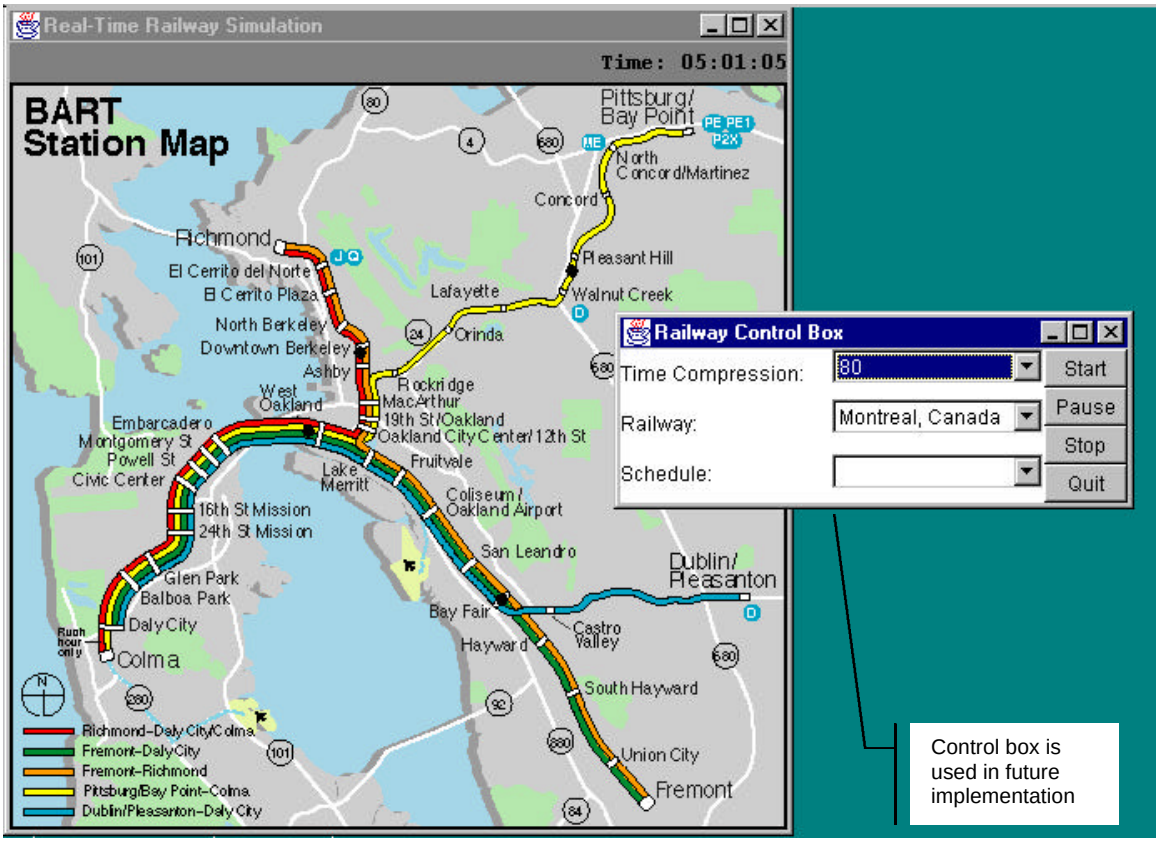



**Figure 1.3 Railway Simulation that is running**

To run the simulation, in the same directory where all the Railway simulation files are, type this line, then press enter on the command line:

```
java RailwaySimulator [name of Railway] [name of Schedule]
```

For example, the railway pictured and included with the Railway Simulation, is run by typing:

```
java RailwaySimulator BART Weekday
```

It will take a while to load the data, so please be patient. In Figure 1.3, the simulation is shown in running mode, for BART in this instance. The figure is inaccurate in showing the Control Box and does not appear in this version, but has been left in for my own personal reasons. Trains will run over each line, which in the above bitmap are shown in different colors (actually, a line is each of the colored lines moving in one direction). In the upper right hand corner, a simulation clock will update frequently, reflecting the time in the simulation.

The time compression is permanently fixed at 40 : 1, which means that for every minute in the railway schedule, 1/40 of a minute, or 1.5 seconds, passes in real life.

The simulation runs until the last train in the schedule completes run. The simulation can be stopped abruptly by clicking on the command line window, the Railway Simulation was started from and typing Control-C.

**Conclusion**
Well, have fun, and wait for an update…

# Future Additions

The following is a list of features desired in a future update of the Real-time Railway Simulation in Java:

- Railway Simulation control box, which allows user to interact with the application with the GUI components in the control box. These components include:
  - Stop, Pause, and Quit buttons
  - Time compression control
  - GUI selection of available railways and associated schedules by users
- Use of more powerful databases, such as Oracle or Informix
- Web browser driven application with JDBC connection over the Internet
- Clicking of objects in Railway animation area, which displays information about those objects, such as ETAs for incoming trains at a station, or originating source of a train, etc.
- Use of Java interfaces for flexibility, when using multiple databases.

# REFERENCE

Here is a list of books used in design Real-time Railway Simulation in Java along with my opinions:

*Booch, Grady*, Object-Oriented Analysis and Design with Applications, 1994, Benjamin and Cummings, Redwood City, CA
An thorough introduction to object oriented design and object-oriented notation.

*Campione and Walrath*, The Java Tutorial, 1996, Addison-Wesley Publishing, Menlo Park, CA
Well written jumpstart on the Java language.

*Flanagan, David*, Java in a Nutshell, 1997, O'Reilly and Associates, Sebastopol, CA
Another introduction to the Java language.

*Gosling, Joy and Steele*, The Java Language Specification, 1996, Addison-Wesley Publishing, Menlo Park, CA
A dictionary for Java on the details of the Java programming language.

*Hamilton, Cattell and Fisher*, JDBC Database Access with Java, 1997, Addison-Wesley Publishing, Menlo Park, CA
A well written introduction to JDBC programming.

*Roman, Steven*, Access Database Design and Programming, 1997, O'Reilly and Associates, Sebastopol, CA
A book on basic database design, E-R diagrams, and MS Access SQL